НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Онтологічна інформаційна система моніторингу показників рівня міжнародного співробітництва»

Виконала
студентка IV курсу, групи ТВ-61
Юрченко Богдана Олегівна _____

Керівник:
Старший викладач
Дацюк О. А. _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____
(прізвище, ім’я, по батькові)

1. Тема роботи __Онтологічна інформаційна система моніторингу показників рівня міжнародного співробітництва_____
_____
керівник роботи_старший викладач Дацюк Оксана Антонівна_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”11” жовтня 2019 р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи __персональний комп’ютер під керуванням операційної системи Fedora Workstation 32 та Xubuntu 20, мова програмування Java, мова програмування JavaScript._____
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)__проаналізувати показники та звіти з міжнародного співробітництва університетів, розглянути існуючі системи для аналізу міжнародного співробітництва, спроектувати інформаційну систему для аналізу та моніторингу міжнародного співробітництва, розробити спроектовану систему.
5. Перелік ілюстративного матеріалу : архітектура проекту, розроблені класи онтології, приклади інтерфейсу програми, приклади роботи із системою_____
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”11” _жовтня_____2019_ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11.10.19	
2.	Вивчення та аналіз задачі	11.10.2019 - 20.01.2020	
3.	Розробка архітектури та загальної структури системи	22.01.2020 - 15.02.2020	
4.	Розробка структур окремих підсистем	16.02.2020 - 01.03.2020	
5.	Програмна реалізація системи	03.03.2020 - 17.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020 - 07.06.2020	
7.	Захист програмного продукту	14.05.2020	
8.	Передзахист	09.06.2020	
9.	Захист		

Студент _____ _____Юрченко Б. О._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Дацюк О. А._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дипломну роботу виконано на 64 аркушах, вона містить 4 додатки та перелік посилань на використані джерела з 12 найменувань. У роботі наведено 40 рисунків.

Метою даної дипломної роботи є створення інформаційної системи для моніторингу показників рівня міжнародного співробітництва університетів.

У роботі було проаналізовано показники міжнародного співробітництва, можливі аналоги для роботи із показниками. Було визначено основні функціональні і не функціональні вимоги до системи.

У процес виконання роботи було спроектовано і розроблено систему для моніторингу показників рівня міжнародного співробітництва. Отриману систему було протестовано.

Ключові слова: міжнародне співробітництво, онтології, показники, OWL.

ABSTRACT

Thesis is done on 64 sheets, it contains 4 appendices and a list of references to the used sources with 12 titles. The paper contains 40 figures.

The purpose of this thesis is to create an information system for monitoring the level of international cooperation of universities.

The indicators of international cooperation, possible analogues for work with indicators were analyzed in the work. The main functional and non-functional requirements for the system were identified.

In the course of the work, a system for monitoring indicators of the level of international cooperation was designed and developed. The resulting system was tested.

Key words: international cooperation, ontologies, indicators, OWL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTML - Hypertext Markup Language;

CSS - Cascading Style Sheets:

PHP - PHP: Hypertext Preprocessor

jar - Java archive;

JSON - JavaScript Object Notation;

XML - Extensible Markup Language;

ООП - об'єктно-орієнтоване програмування.

owl - Web Ontology Language

RDF - Resource Description Framework

SPARQL - SPARQL Protocol and RDF Query Language

API - Application Programming Interface

REST-API - Representational State transfer Application Programming Interface

IDE - Integrated Development Environment

SPA - Single Page Application

DI - Dependency Injection

HTTP - Hypertext Transfer Protocol

ОС - операційна система

OIDC - OpenID Connect

JWT - JSON Web Token

ВСТУП

Правильний аналіз діяльності роботи деякого закладу - це одна із найважливіших умов його подальшого існування і розвитку. Адже стратегія майбутніх дій може бути побудована лише на основі правильно зроблених висновків про колишні проблеми, рішення і наслідки.

Міжнародна діяльність є дуже важливою і нелегкою частиною діяльності навчальних закладів. Кожен вуз прагне давати найкращі можливості своїм співробітникам і студентам, а також бути відомим у світі. Кожного року проводиться аналіз міжнародного співробітництва і складається звіт, який сприяє кращим змінам у майбутньому.

Міжнародне співробітництво - це різні типи співпраці організацій або окремих людей. Це насамперед дуже багато інформації, яку треба якимось чином структурувати, і тому задача менеджменту такої інформації на сьогодні є дуже важливою.

Мета даної роботи - розробити онтологічну систему, що полегшить співробітникам університетів роботу і дозволить проводить аналіз міжнародної діяльності простіше й ефективніше.

Постановка задачі виникла в рамках науково-дослідної роботи на тему: «Дослідження і впровадження ключових технологій для моніторингу розвитку міжнародного співробітництва та створення системи підтримки ухвалення рішень в науково-технічній сфері».

У першому розділі визначаються поставлені завдання. У другому розглядається предметна область : показники міжнародної діяльності. У третьому - архітектура програмного рішення. В останньому розділі подається інструкція до розробленої системи.

1. ЗАВДАННЯ РОЗРОБКИ ІНФОРМАЦІЙНОЇ ОНТОЛОГІЧНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ МІЖНАРОДНОГО СПІВРОБІТНИЦТВА

Мета даної роботи - розробити інформаційну систему (програмний продукт), що змогла б спростити задачі збирання, зберігання, управління та аналізу інформації про міжнародне співробітництво вищих навчальних закладів. Для досягнення даної мети мають бути виконані такі завдання, як: розглянути задачі аналізу і підготовки звітів про міжнародне співробітництво вищих навчальних закладів. Проаналізувати показники, що використовуються і обраховуються. Спроектувати модель даних, що б дозволила в достатній мірі детально і гнучко описувати показники міжнародного співробітництва.

Задача роботи була визначена в результаті роботи на тему: «Дослідження і впровадження ключових технологій для моніторингу розвитку міжнародного співробітництва та створення системи підтримки ухвалення рішень в науково-технічній сфері». Завданнями роботи є дослідження і розробка ключових технологій для створення міжнародної системи моніторингу інформації та підтримки прийняття рішень в галузі міжнародного науково-технічного співробітництва вздовж «Одного поясу, одного шляху», і пропозиція платформ для її впровадження, забезпечуючи авторитетні і ефективні консультації державним установам і підприємствам при прийнятті рішень і інвестуванні, надання точних та інформативних даних для наукових досліджень і міжнародного науково-технічного співробітництва.

Спроектувати і розробити програмне забезпечення, що дозволило б керувати даними про міжнародне співробітництво. Підготувати програмний продукт до встановлення і описати процес розвертання.

Програмний продукт має відповідати таким основним нефункціональним вимогам:

- система має бути простою у використанні і не вимагати довгого установлення і налаштуванням перед початком роботи;

- система повинна мати зручний зрозумілий інтерфейс;
- система має захищати дані від несанкціонованого доступу;
- система має бути платформонезалежною.

Система має бути спроектована таким чином, щоб дозволити легке розширення у майбутньому (розробка інших клієнтів, наприклад мобільного додатку тощо).

До основної функціональних вимог входять такі:

- система повинна мати механізм авторизації і аутентифікації;
- система повинна мати декілька ролей користувачів із різним доступом до даних;
- система повинна дозволяти вводити, редагувати та видаляти дані;
- система повинна дозволяти керувати користувачами (створювати, редагувати, видаляти);
- система повинна дозволяти переглядати дані у 2 режимах;
- система повинна дозволяти керувати різними базами даних показників міжнародного співробітництва;
- система повинна дозволяти імпортувати, експортувати дані у файлах із розширенням .owl;
- у системі повинна бути реалізована функція створення версій;
- система має дозволяти працювати із різними базами даних одночасно і переключатись між ними;
- система повинна дозволяти одночасно працювати декільком людям із одними даними.

Більш конкретно, система повинна дозволяти вводити дані про різні показники міжнародної співпраці вузів, програми, структуру ВНЗ, досягнення і тому подібне. У користувачів має бути можливість редагувати дані, переглядати у вигляді таблиць і робити запити до системи з метою аналізу даних.

Система повинна приймати на вхід обраховані значення різних показників міжнародної співпраці університетів. Система повинна уміти працювати із даними у

форматі owl, тобто із онтологіями із визначеними класами і властивостями, збереженими у синтаксисі “RDF/XML”.

У системі повинна бути реалізована функція виконання запитів на мові запитів SPARQL (лише для перегляду даних, а не їх модифікації). Також у системі повинен бути присутній інший режим перегляду даних - у вигляді таблиць.

Робота із системою має проходити в онлайн режимі. Різні користувачі повинні мати можливість працювати із одними даними одночасно (таким чином система має бути реалізована у формі веб-додатку). У системі має бути функція вибору набору даних, із якими працює користувач.

Система повинна дозволяти працювати із різними базами (наборами) даних та їх версіями. Система повинна забезпечувати різні рівні доступу до даних на основі ролей користувачів. Система повинна мати 3 ролі користувачів.

Система повинна мати зручний сучасний користувацький інтерфейс.

1.1. Особливості розробки web-клієнтської частини програмного рішення

Для реалізації усіх загальних вимог до системи (наприклад, можливість роботи декількох користувачів над одним проектом) було вирішено розробляти систему у вигляді веб-додатку. Через це до реалізації різних модулів програмного продукту накладаються додаткові вимоги.

Web-додаток повинен бути розрахований на 2 категорії користувачів і також на 3 ролі. Так користувач може бути незареєстрованим в системі і зареєстрованим. Система повинна уміти працювати із зареєстрованими користувачами, що мають одну або більше ролей :

- користувач із можливістю перегляду даних (r_user);
- користувач із можливістю редагування даних (rw_user);
- користувач-адміністратор (admin).

Система повинна бути максимально закритою. Це означає, що вона повинна якнайбільше обмежувати доступний функціонал для незареєстрованих користувачів. Таким чином у незареєстрованих користувачів повинен бути доступ лише до головної сторінки, яка містить лише інформацію про основні можливості системи.

Також система не повинна дозоляти користувачам самостійно реєструватись. Усі завдання із надання доступу повинні виконуватись спеціальним користувачем, що має на це права. Це роблять адміністратори системи, що мають реєструвати усіх користувачів і видавати їм дані для доступу. Таким чином у системі не повинно бути сторінки реєстрації. Має бути присутньою лише сторінка авторизації (за допомогою логіну і паролю).

Система повинна обмежувати доступні для користувача елементи управління і сторінки на основі ролей користувача. Таким чином кожен із користувачів не повинен мати змоги переходити до сторінок, що не є доступними для ролі із найбільшим рівнем доступу користувача .

1.2. Користувацький інтерфейс системи

Система повинна відповідати сучасним стандартам користувацьких інтерфейсів. Тобто повинна мати чистий, функціональний, зрозумілий дизайн. Вона не повинна бути переобтяженою функціями, що не є основними під час роботи із показниками міжнародної діяльності.

Усі елементи управління мають виділятися і не повинні бути прихованими, тобто система повинна бути інтуїтивно зрозумілою.

Система повинна повідомлювати про усі успішні операції модифікації даних. Так само, програма повинна повідомляти користувачеві про усі помилки, що виникли під-час роботи. Система не має дозволяти робити випадкові модифікації даних (наприклад випадкове видалення тощо)

Усі елементи управління повинні мати необхідні розміри для і елементи управління для зручного введення даних.

1.3. Вимоги до розробки REST-API

Back-end частина системи повинна реалізувати усі дії, зв'язані безпосередньо із обробкою даних. Back-end частина має бути реалізована у формі REST-API. Це означає, що клієнтська частина повинна спілкуватись із серверною частиною за допомогою запитів із використанням JSON.

API повинне відповідати сучасним вимогам проектування і створення API а також бути інтуїтивно зрозумілим. Кожен енд-поінт API повинен мати налаштовані конфігурації доступу (на основі ролей користувача).

1.4. Потенційні користувачі

Система, що розробляється, в першу чергу орієнтована на людей, що займаються аналізом міжнародного співробітництва університетів. Частина, що робить систему орієнтованою на міжнародне співробітництво - це розроблена модель даних (онтологія).

Також система дозволяє завантажувати свої онтології і працювати із ними. Тому загалом система може бути використана для редагування будь-якої онтології і буде корисною кожному, хто не має навичок роботи з більш професійними інструментами розробки і має потребу вводити дані в онтології.

Таким чином мета роботи - розробити інформаційну систему, що ідеально налаштована і підходить для роботи із різними показниками міжнародної діяльності університетів (зберігання, модифікація та аналіз). Програмний продукт може працювати також із іншими онтологіями (що не поставляються разом із розробленим програмним забезпеченням). Таким чином розроблена система також буде цікава і корисна тим людям, що мають мінімальний досвід роботи.

2. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ: ПОКАЗНИКІВ МІЖНАРОДНОГО СПІВРОБІТНИЦТВА

Кожного року університети складають звіти про результати міжнародної діяльності. У звітах підраховується й аналізується усе зроблене за рік. У звітах підсумовується виконання завдань, що були поставлені. Також визначається, чи була виправдана стратегія, обрана до цього, чи треба якось її змінити і куди рухатись далі.

Дані звіти можна знайти на офіційних сайтах вищих навчальних закладів. Під час роботи було проаналізовано звіти про міжнародну діяльність КПІ[1][2], Київського Міжнародного Університету[3], Ужгородського Національного Університету[4], Університету Банківської Справи[5] та деяких інших.

2.1. Результати аналізу звітів про міжнародне співробітництво різних університетів

При аналізі міжнародного співробітництва обраховують такі показники:

- обсяг міжнародних контактів,
- кількість міжнародних заходів, таких як конференції, форуми, семінари тощо
- кількість візитів делегацій
- кількість фахівців у складі делегацій
- кількість виїздів співробітників
- кількість іноземних студентів
- викладачів (за науковими ступенями)[2].

Аналізується співпраця з різними організаціями, укладені договори, розпочаті і закінчені проекти. Участь університету у міжнародних програмах: наприклад Erasmus+, тощо.

Усі показники рахуються як для університету в цілому, так і для окремих його підрозділів (наприклад за факультетами та кафедрами, лабораторіями). Це потрібно, щоб визначити рейтинг підрозділів за різними показниками. Не рідко усі показники також визначаються за спеціальностями і спеціалізаціями : наприклад кількість написаних робіт за якоюсь спеціальністю або кількість іноземних викладачів, що викладають предмети якоїсь спеціальності. Також показники, де підсумовується кількість викладачів можуть бути поділені для визначення кількості викладачів із кожним із наукових ступенів.

У даних звітах також аналізуються витрачені кошти і їх надходження. Також підсумовується, на що кошти було витрачено : на програми, обладнання тощо.

Також для більшої наглядності можуть порівнюватись значення тих самих показників за минулі роки.

Таким чином кожен показник за своєю суттю є числом але при цьому кожен показник має своє значення (це кількість викладачів, програм чи коштів). Кожен показник рахується за якийсь час чи період, кожен показник обраховується для якогось підрозділу університету чи університету у цілому. Кожен показник повинен може мати додатковий опис (назви робіт, опис важливості показнику чи простий коментар).

Таким чином написання звіту про міжнародну діяльність вимагає неймовірної кількості роботи, уваги, організованості та пам'яті. Найбільш головна проблема у зберіганні значень показників, адже вони можуть обраховуватись для різних значень аргументів. Показники також можуть бути використані для порівняння через деякий час (рік або кілька років тощо). Це означає, що кожного року показники мають бути обраховані для усіх можливих поєднань значень цих аргументів (для можливості порівняння у майбутньому). Для того, аби уникнути складних обрахунків кожного разу потрібно пожертвувати іншим ресурсом - пам'яттю. Тобто зберігати окремі записи про кожну подію, аби в майбутньому можна було обрахувати значення показників із потрібною детальністю. В такому разі у людини, що складає звіт також мають бути хороші навички написання запитів до бази даних.

Аналіз існуючих звітів із міжнародного співробітництва показав, що предметна область неймовірно обширна і містить у собі декілька дерев об’єктних типів і об’єктів (у разі реалізації із зберіганням лише обрахованих значень показників). Дерево об’єктних типів показників може нараховувати 50-100 класів і може розширюватися у майбутньому. Це означає, що система, що розробляється потребує модель даних, що дозволяє елегантно і просто описувати ієрархії об’єктниї типів і що дуже легко розширяється. Саме розробка такої моделі і вибір бази даних і є найважливішою задачею.

Таким чином головною задачею програмного продукту, що розробляється є спрощення написання запитів із урахування зв'язків агрегації і наслідування. А також гнучке зберігання об’єктів та значень властивостей цих об'єктів і забезпечення просторо розширення об’єктних типів безпосередньо після початку користування програмного продукту.

Таким чином була проаналізована предметна область показників міжнародної діяльності університетів і значення складених звітів про діяльність для адміністрації університетів. Такі звіти допомагають проаналізувати роботу університету, визначити головні помилки та розробляти план і стратегію роботи на наступні роки. Були визначені головні складності розробки програми для вирішення проблем аналізу значень даних показників.

2.2. Опис існуючих аналогів

Наразі аналогів систем для аналізу міжнародного співробітництва вузів у вільному доступі не існує.

Звичайно для зберігання інформації і її переглядання можна використовувати будь-яке із безлічі готових рішень на ринку (і тих, що не розроблені спеціально для даної предметної області). Наприклад пакет для офісу Microsoft : Excell, Access; реляційні бази даних із простими інтерфейсами : MySql із phpMyAdmin. Але такі рішення не є ідеальними. Наприклад використання простої бази даних і інструменту

для адміністраторів вимагають значних знань про реляційні бази даних та їх управління. Також проблема із ідеальною схемою бази даних не вирішена, адже реляційні бази даних не є ідеальним рішенням для реалізації ієрархій об'єктних типів.

Використання програмних продуктів від Microsoft має іще більше проблем :

- зберігання даних локально (в одному місці)
- неможливість працювати одночасно
- залежність від програмного забезпечення, (аби відкрити таку "базу даних" треба встановити на робочу машину ту ж саму версію програми, що і та, в якій файл був відредагований). В противному випадку можливе некоректне відображення даних, їх втрата тощо.
- створення резервних копій.

Але головною проблемою такого використання програм Microsoft є неможливість оперування даними за допомогою запитів.

Отже наразі на ринку немає програмного забезпечення для вирішення задачі аналізу показників міжнародного співробітництва вузів у вільному доступі. Це означає, що дане дослідження на даний момент є унікальним і розроблене рішення - онтологічна інформаційна система може не мати аналогів.

3. ЗАСОБИ РОЗРОБКИ

Під час розробки системи були використані різні середовища розробки для різних частин системи. Так для сервер-частини веб-додатку було використане інтегроване середовище розробки - Intelij Idea. Для розробки користувацького інтерфейсу інший продукт JetBrains - Webstorm - інтегроване середовище розробки, що націлене саме на побудову додатків для браузеру. Воно чудово працює із найпопулярнішими фреймворками і, так само як і інші продукти від JetBrains , має неймовірну кількість плагінів. Усі продукти JetBrains доступні безкоштовно для студентів та викладачів університетів.

Під-час розробки активно використовувалась система контролю версій - git, що на сьогодні є найпопулярнішою.

Для розробки онтології показників міжнародного співробітництва було використане Protege - розробка Стендфорського університету.

Для деплою різних частин системи був використаний Docker. І для розміщення контейнерів в інтернеті - був використаний хостинг для контейнерів - sloppy.io.

3.1. Інтегровані середовища розробки - Intelijj idea та webstorm

Для розробки серверної частини програмного продукту булу використане середовище розробки - IntelliJ IDEA. Це середовище програмування (IDE), орієнтоване на розробку на мові програмування Java, розроблене компанією JetBrains. Програмний продукт доступний у трьох виданнях: Community Edition, що має ліцензію Apache 2.0, комерційне видання, відоме як Ultimate Edition, а також Education Edition. IntelliJ IDEA вважається одним із найкращих IDE на даний момент.

WebStorm - це іще одна розробка JetBrains. Він дуже схожий на IntelliJ IDEA. Такий самий дизайн, основні функції - це автодоповнення, пошук, потужні засоби

для рефакторингу, підтримка безлічі плагінів. Але дане середовище націлене саме на розробку front-end частин веб-додатків.

3.2. Середовище розробки онтологій - Protégé desktop

В якості сховища даних було вирішено використовувати онтології. Завдяки своєму представленні у вигляді графів вони є хорошим рішенням головної задачі розробки системи для аналізу міжнародного співробітництва.

Онтології в інформатиці - це формалізоване представлення про деяку предметну область. Зазвичай їх використовують для опису термінів і їх зв'язків. На сьогодні вони використовуються у великих інтернет-магазинах, наприклад для збереження інформації про продукцію, що продається. І хоча пік популярності онтологій припав на минулі роки, однак і зараз активно розробляються і підтримуються системи розробки онтологій. Безперечно, найкращим рішенням, що зараз є доступним, - це рішення Стендфортдсього університету -Protege (рисунок 3.1).

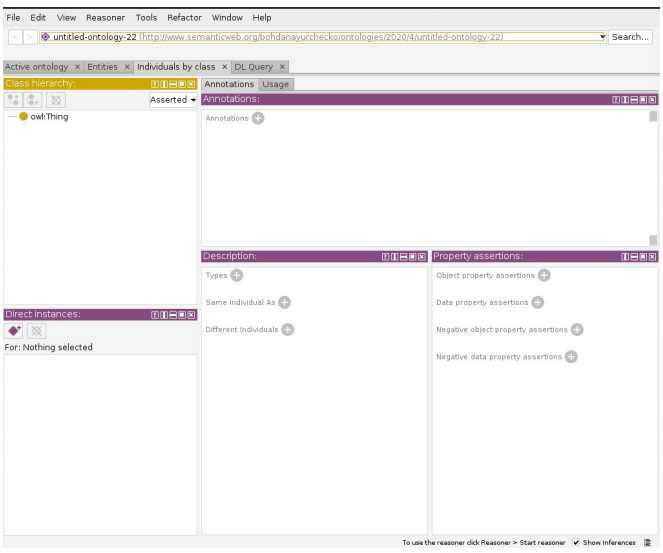


Рисунок 3.1 - Інтерфейс програми Protege

Protégé - це безкоштовна платформа з відкритим кодом, що забезпечує зростаючу спільноту користувачів набором інструментів для побудови доменних

моделей та знань на основі знань із онтологіями. На сьогодні доступна для завантаження версія 5.5.0 для Windows, Linux, а також платформонезалежна версія. Даний програмний продукт написаний на Java, що робить його універсальним.

Десктоп версія є ідеальним рішенням для професіоналів, яким потрібен потужний інструмент для розробки. Вона дуже легко розширяється за допомогою плагінів. Має підтримку OnTop для зв'язку онтології і бази даних. Підтримує багато різних різонерів. І загалом має усі можливі функції, але є переобтяженою і незрозумілою для новачків.

Також доступна Web-версія, яку не потрібно встановлювати на робочу машину для початку роботи (рисунок 3.2). Вона має більш чистий сучасніший інтерфейс. Також є можливість спільної роботи над одним проектом, що є великим плюсом. Але програмний продукт все іще націлений на людей, які професійно займаються розробками онтологій.

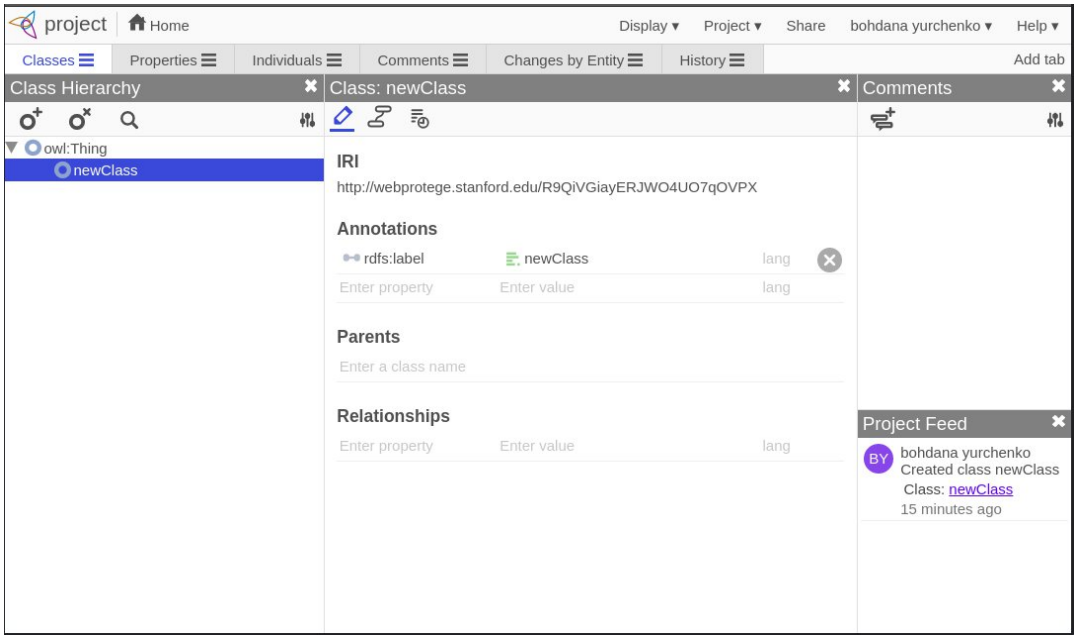


Рисунок 3.2 - Інтерфейс Web-додатку Protege

3.3. Angular фреймворк для розробки front-end частини

Для розробки браузерної частини додатку використовували фреймворк - Angular 8. Angular підтримується Google з'явився на ринку іще у 2009 році. Таким

чином це перевірене часом рішення, що стало іще більш популярним із роками, що має велике ком'юніті і чудову підтримку.

Для додавання до програми матеріал дизайну був використаний пакет angular/material. Angular Material - це бібліотека компонентів інтерфейсу для розробників Angular. Компоненти Angular Material допомагають створювати привабливі, послідовні та функціональні веб-сторінки та веб-додатки, дотримуючись сучасних принципів веб-дизайну.

На рисунку 3.3 зображений приклад розмітки хедеру додатку із використанням компонентів із бібліотеки матеріального дизайну.

```
<mat-toolbar color="accent">
  <h1 class="mat-display-1" (click)="router.navigate( commands: ['/']);" style="...">Ontology App</h1>
  <span class="example-spacer"></span>
  <button mat-button (click)="selectOntology()"
          *ngIf="(userInfo | async)?.isLoggedIn">{{selectedOntologyFile ? selectedOntologyFile : "Not specified"}}
  </button>
  <button mat-button routerLink="/class-tree-component"
          *ngIf="(userInfo | async)?.isLoggedIn">Class Tree
  </button>
  <button mat-button routerLink="/query-page"
          *ngIf="(userInfo | async)?.isLoggedIn">
    Query Page
  </button>
  <button mat-button routerLink="/admin-page"
          *ngIf="(userInfo | async)?.isLoggedIn && (userInfo | async)?.isAdministrator">Admin Page
  </button>
  <button mat-button [matMenuTriggerFor]="belowMenu"
          *ngIf="(userInfo | async)?.isLoggedIn; else templateForNotLoggedIn">{{ (userInfo | async)?.username }}
  </button>
```

Рисунок 3.3 - Приклад розмітки хедеру сайту

Таким чином у інтерфейсі програмного продукту дуже часто були використані компоненти із бібліотеки Angular Material. Такі, як наприклад, різні кнопки, форми, таблиці. Компоненти цієї бібліотеки дозволили дуже просто побудувати функціональний і гарний користувацький інтерфейс.

Такі особливості Angular як підтримка ООП, використання тайп-скріпту, впровадження залежностей [11], реактивність тощо дозволили дуже легко і просто розробити клієнтську частину системи, яку в майбутньому буде легко підтримувати і розширювати.

3.4. Apache Jena - Java бібліотека для роботи із онтологіями

Для реалізації роботи із онтологіями на сервер частині системи була використана бібліотека Apache Jena. Apache Jena - це вільна та відкрита бібліотека для Java для створення семантичних веб-додатків та додатків із пов’язаними даними.

Основними функціями, що були використані в системі - це читання моделі із файлу .owl, запис моделі у файл, а також зчитування, конструювання та запис класів, інтстансів та властивостей онтології.

Бібліотека також має модуль для виконання SPARQL[9] запитів, у тому числі із використанням різних різонерів.

3.5. Docker - інструмент для створення і менеджменту контейнерів

Усі частини програмного забезпечення упаковуються у контейнери для запуску із допомогою Docker та Docker Compose. Контейнер у програмуванні - це стандартний блок програмного забезпечення, який пакує код та всі його залежності, тому програма швидко та надійно може бути перенесена з одного обчислювального середовища в інше. Docker - це найбільш популярний інструмент для контейнеризації додатків.

Контейнери дозволяють переносити програми із усіма встановленими залежностями між середовищами а також ізолювати різні програми, що знаходяться на одній машині оду від одної. Це приносить неймовірно багато плюсів, серед яких - проста установка програм, захист програм, економія ресурсів і найголовніше - це економить витрати, в тому числі і на обслуговування обладнання. Економія коштів робить контейнери ідеальним рішенням в більшості випадків, у тому числі і для даного проекту [10].

Docker Compose - це інструмент для визначення і запуску додатків, що складаються із багатьох контейнерів (більше одного). Розроблений продукт саме такий : складається із 3 модулів. Docker Compose робить використання Docker іще

простішим - адже дозволяє виконувати дуже багато операцій, що потрібні для запуску мульти-контейнерних додатків, за допомогою лише однієї команди.

Процес запуску розробленого програмного забезпечення за допомогою Docker Compose і Docker будуть детально розглянуті в останньому розділі - “Робота користувача з програмною системою”.

3.6. Хостинг для Docker -контейнерів - sloppy.io

У якості хостингу Docker -контейнерів під час розробки був обраний sloppy.io. Це хороший хостинг із зручним інтерфейсом, усіма потрібними інструментами для деплою і моніторингу контейнерів. Має зручні і прості засоби конфігурації (CLI та WEB UI), а також декілька планів з різними можливостями і за різними цінами, а також безкоштовний термін для випробування перед оформленням повноцінного плану.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Для реалізації системи була обрана форма веб-додатку. У якості сховища даних були обрані онтології. Для реалізації авторизації та аутентифікації і рівнів доступу до функцій програми та даних був обраний Keycloak. Для роботи з онтологіями та RDF файлами була обрана безкоштовна бібліотека із відкритим кодом - Apache Jena.

У рамках даної роботи система була спочатку спроектована. На рисунку 4.1 зображено схему модулів програмного продукту. На схемі видно основні модулі програми та їх взаємодія.

Таким чином деякі модулі клієнтської частини взаємодіють безпосередньо із користувачем. Як на клієнтській частині так і на серверній частині є модулі для обробки http запитів. На серверній частині є модуль роботи із файловою системою - для роботи із файлами онтологій, що знаходяться в системі.

Зі схеми видно, що система може працювати із декількома онтологіями. Але в рамках даної роботи було розроблено одну онтологію, що дозволяє зберігати значення показників міжнародного співробітництва. Детальніше розроблена онтологія описана у розділі 4.2. Саме ця онтологія поставляється із програмою.

Уся обробка даних проходить на серверній частині системи. Клієнтський додаток відповідає за відображення даних та взаємодію із користувачем.

Через деякі особливості онтологій - форма для створення об’єктів є унікальною для кожного класу онтології. Саме тому у системі виникла потреба у модулі для побудови форм.

Через те, що під час виконання деякої операції можуть виникнути помилки і ці помилки мають бути передані користувачеві у простій і зрозумілій формі - був розроблений модуль для оброблення помилок.

Детальніше про архітектуру системи, функції модулів та пакетів і приняті рішення розказано у наступних підрозділах.

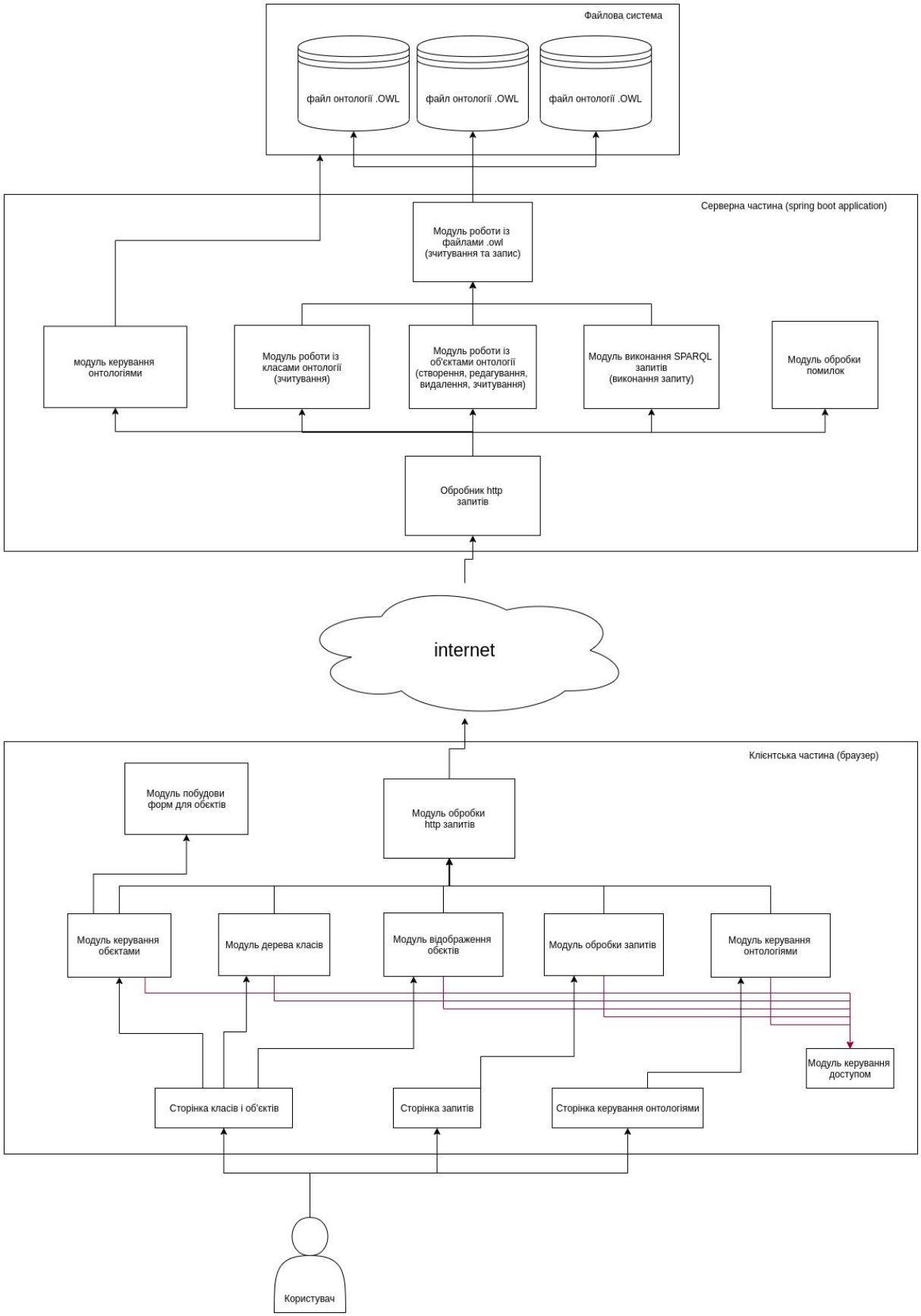


Рисунок 4.1 - Схема модулів програмного продукту

4.1. Вибір онтології в якості сховища даних

Для реалізації даного програмного продукту для аналізу міжнародного співробітництва були обрані онтології за такими причинами:

- Предметна область є дуже обширною. Вона має неймовірну кількість різних сутностей і усі вони мають різні зв'язки між собою.

У разі вибору реляційної бази даних в якості сховища - підготовка і проектування бази даних вимагало б у рази більше часу. Також реалізація із реляційною базою даних мала б передбачити усі можливі точки розширення системи. Адже зміна схеми реляційної бази даних уже під час експлуатації системи може привести до серйозних наслідків (у тому числі і до унеможливлення роботи із системою через деякий час).

- Дані сутності можуть змінюватися, їх зв'язки можуть змінюватись теж.
- Сутності, що аналізуються, поділяються на декілька дерев (дерев класів і об’єктів).

Реалізація ієрархії класів об'єктів у базі даних - це завжди не найелегантніше рішення. Зазвичай використовуються різні таблички для кожного підкласу або поля, які можуть бути пустими або заповненими в залежності від класу запису в таблиці. У такому разі також використовуються різні поля-флажки. Як правило, їх логіка не очевидна і в майбутньому вони можуть стати проблемою.

Цю задачу також можна вирішити за допомогою мета-моделі бази даних. У такому разі метадані даних, що є в базі даних, зберігаються як окремі записи в окремих таблицях в самій базі даних. Такий метод передбачає існування таких таблиць в базі даних, як “Тип”, “Клас”, “Об’єкт”, “Властивість”, “Значення Властивості”. Дана схема є неймовірно універсальною і розширюваною (адже в такому випадку метадані можна міняти без зміни схеми бази даних, тобто зі зміною в поведінці сутностей в предметній області не потрібно міняти схему бази даних).

Мінусом цього підходу є те, що працювати із такою базою даних важко. Також запити на мові SQL значно збільшуються (якщо порівнювати із запитами до баз даних, реалізованих звичайним підходом).

Тому для реалізації даної інформаційної системи для зберігання даних потрібно було сховище, що :

- дозволило б елегантно описувати ієрархії класів сутностей предметної області
- дозволило б змінювати метадані класів безпосередньо під час експлуатації системи
- було простим в експлуатації і дозволило б писати прості і короткі запити.

Онтології підходять за усіма вимогами, тому вони і були обрані в якості сховища даних.

4.2. Розроблена онтологія для зберігання показників міжнародного співробітництва

На основі аналізу існуючих звітів про міжнародну діяльність була розроблена онтологія, що дозволить заносити дані про усі потрібні показники, що аналізуються (рисунок 4.2).

В онтології було додано такі класи, як :

- Країна
- Наука
- Науковий ступінь
- Показник
- Проект
- Організація.

Основним класом є клас “Показник”, який має різні більш конкретні підкласи. Кожен показник за своєї суттю - це число. Кожен показник має також поле із типом “dateTime” що дозволить зберігати значення показників за конкретні дати чи роки. Також кожен показник має поле “Коментар”, де можна у вільній формі залишити

більш конкретний опис значення показника. Наприклад, це потрібно для показника “Обсяг благодійної допомоги”: значення - це кількість грошей, а в коментар можна написати, на що вони були витрачені.

Рисунок 4.2 - Дерево класів розробленої онтології

Деякі показники мають прив’язку до програми (наприклад кількість студентів, що взяли участь у програмі, кількість викладачів, що взяли участь у програмі). Тому в онтологію був доданий клас “Програма” із різними підкласами.

У звітах також аналізують співпрацю університетів із іншими країнами, або програмами, що є міжнародними або закордонними. Для збереження цієї інформації в онтологію було додано клас “Країна”. А кожен проект має такі властивості. як “заснований” і “країна_учасник” - посилання на екземпляри класу “Країна”.

Клас “Наука” і підклас “Спеціальність” потрібні для збереження значень показника за спеціальностями.

Різні підкласи класу “Організація” : “Університет”, “Факультет”, “Кафедра”, “Лабораторія” потрібні для збереження значень показників для різних частин університету так само, як і для усього університету в цілому.

Скріншот основних класів онтології та їх підкласів зображено на рисунку 4.1.

Таким чином була розроблена онтологія із усіма потрібними деревами класів для простого початку роботи із міжнародним співробітництвом університетів. Один із плюсів використання онтології в тому, що у майбутньому буде дуже просто додати класи, яких не буде вистачати або змінити уже існуючі.

4.3. Use-Cases системи . Ролі користувачів

Була розроблена діаграма прецедентів, що описує увесь необхідний функціонал системи.

Програмний продукт використовує ролі користувачів для визначення прав кожного користувача, що працює із системою (рисунок 4.3). Цими ролями є :

- r_user (користувач із правом перегляду інформації);
- rw_user (користувач із правом модифікації інформації);
- admin (адміністратор системи).

Користувачі, що мають лише роль “r_user” можуть лише переглядати дані у вигляді таблиць і робити запити на мові SPARQL, що не модифікують дані.

Користувачі, що мають роль “rw_user” можуть , окрім дій, що є доступними для ролі “r_user”, також вводити і редагувати дані (створювати, редагувати і видаляти інстанси із онтологій).

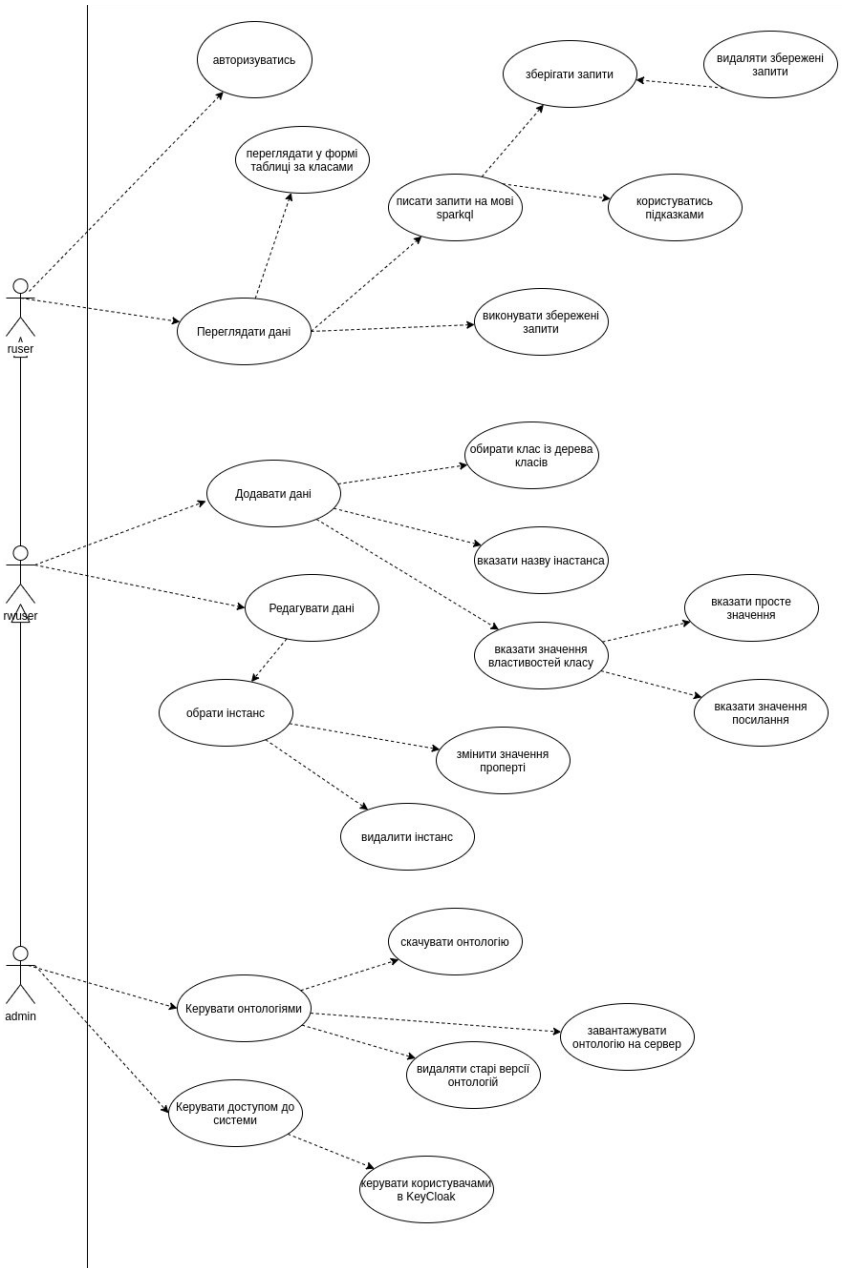


Рисунок 4.3 - Use-Cases розробленої системи

Користувач із роллю “admin” має усі права, що й користувачі із попередніми ролями, плюс додаткові : завантажувати, вивантажувати та видаляти із системи файли із онтологіями (файли у форматі “.owl”).

Усі дані можливості і ролі, яким вони доступні, більш детально описані на діаграмі use-cases. На діаграмі користувач “admin” розширяє користувача “rw_user”, який в свою чергу розширює роль “r_user”. Це означає, що кожній ролі із більшим рівнем доступу доступні усі функції, що має роль із меншим рівнем доступу.

4.4. Реалізація клієнтської частини - web-додатку

Було вирішено реалізувати дану систему у формі односторінкового веб-додатку. Така архітектура є найкращим вибором на сьогодні і має ряд переваг над попереднім підходом - багатосторінкових додатків.

Односторінкові додатки працюють в браузерах з метою забезпечити користувачу досвід, близький до користування десктоп програмою. Робота із SPA не лише дуже схожа на роботу із настільним додатком, а й має переваги над ним: SPA працює в браузері. Це означає, що додаток не вимагає інсталяції перед початком використання. Також це означає, що робота із веб-додатком не залежить від операційної системи і налаштувань робочої машини.

Веб-додаток складається із 2 частин, що працюють поруч (рисунок 4.4). Це:

- Код на стороні клієнта - код, який знаходиться в браузері і відповідає на деякий ввід користувача (написаний із використанням HTML, CSS, JavaScript)
- Код на стороні сервера - код, який знаходиться на сервері і відповідає на HTTP-запити (Java, .NET, PHP, Python тощо).

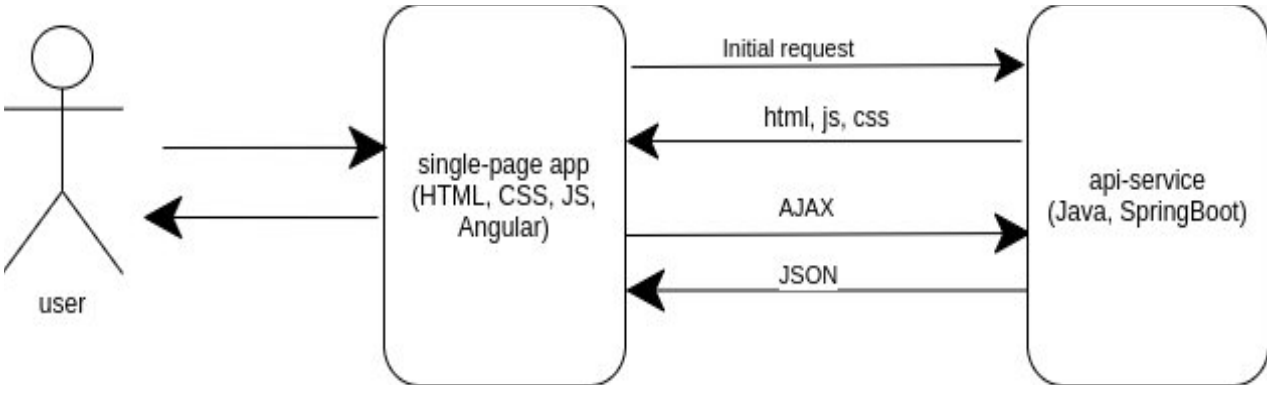


Рисунок 4.4 - Схема взаємодії компонентів SPA

На рисунку представлена схема взаємодії двох частин додатку для класичного SPA веб додатку. Клієнтський код , що працює у браузері у користувача, зв’язується із серверною частиною лише через HTTP-запити. Іще одним плюсом даного підходу є те, що в майбутньому уже розроблена сервер-частина може бути використана повторно під час розробки іншого клієнта (наприклад андроід додатку, тощо).

На рисунку 4.5 зображену структуру проекту клієнтської частини розробленого додатку.

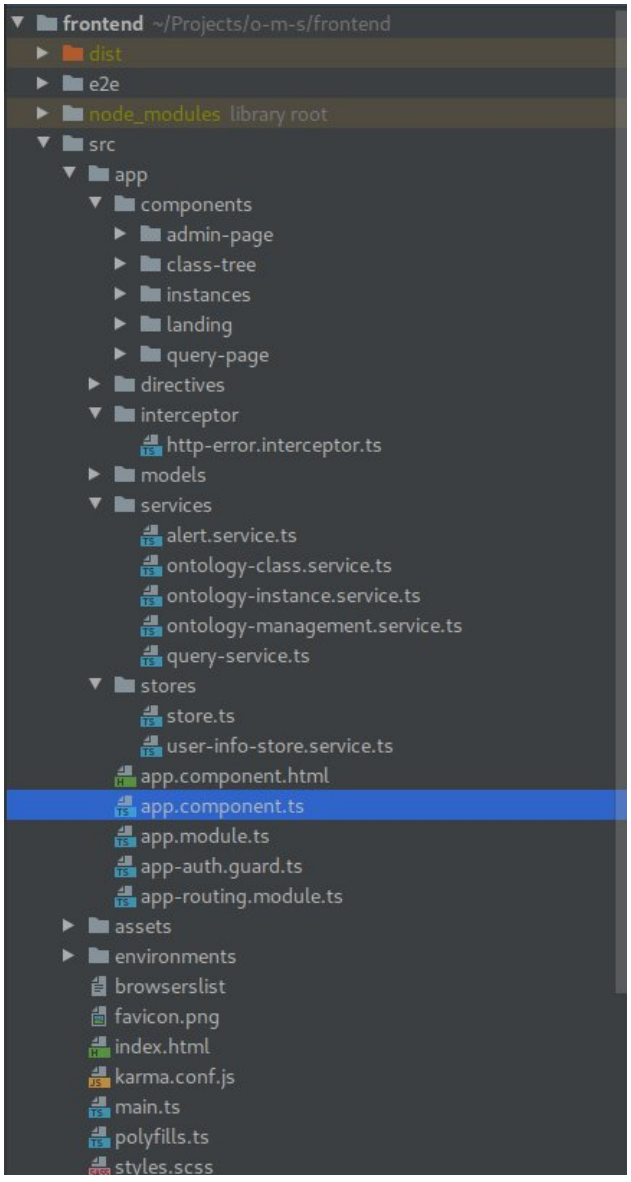


Рисунок 4.5 - Структура проекту

Проект складається із таких пакетів, як components, directives, interceptor, models, services, stores. У пакеті components розміщуються усі не головні компоненти додатку - це сторінки і деякі елементи сторінок (наприклад дерево класів). Усі класи в цьому пакеті відповідають лише за відображення. Дані тут не мають оброблятись.

Уся обробка даних проходить у сервісах, що знаходяться у пакеті services. Усі сервіси є Injectable класами. Це дозволяє користуватися DI фунцією Angular .

Сервіс, що знаходиться в пакеті iterceptor відповідає за перехоплення і обробку усіх помилок. Класи із пакету stores використовуються для збереження і використання інформації про активного користувача. У пакеті models знаходяться усі класи, що описують сутності, якими оперує веб-додаток.

Отож для реалізації клієнтської частини додатку і користувацького інтерфейсу була обрана архітектура односторінкового додатку. Для розробки був використаний фреймворк - Angular . Він дозволяє будувати гарні, потужні додатки, використовуючи усі переваги DI (Dependency injection) та об’єктно орієнтованого програмування.

4.5. Основні логічні елементи клієнтської частини

На основі розробленої діаграми прецедентів, був спроектований інтерфейс. Таким чином веб-додаток складається із декількох сторінок:

- головна;
- сторінка із деревом класів (“Class Tree”);
- сторінка для розробки запитів (“Query Page”);
- сторінка для менеджменту онтологій (“Admin Page”).

Головна сторінка є доступною як для зареєстрованого користувача, так і для незареєстрованого. Головна сторінка описує базові можливості програмного продукту і має посилання на інші сторінки.

Сторінка із деревом класів дозволяє переглядати класи, що є наявними у онтології, а також інстанси із онтології за класами або усі підряд. Ця сторінка також дозволяє створювати, редагувати та видаляти інстанси. Звичайно, цю сторінку

користувачі із різними ролями бачать по-різному. Так, наприклад, користувач, у якого є лише роль “r_user”, не зможе зробити ніяких дій, що змінять дані в онтології. Він зможе лише переглядати дані.

Сторінка для розробки запитів складається із 2 панелей: для введення запиту і для перегляду відповіді. Користувач, що має тільки роль “r_user” зможе виконувати лише запити, що повертають дані, але не модифікують їх.

Сторінка для менеджменту онтологій доступна лише для користувачів із роллю “admin”. Тут є список усіх онтологій, що присутні у системі. Користувач може їх вивантажувати на робочу машину, редагувати і завантажувати назад. Також адміністратор може видаляти із системи старі онтології або версії онтологій.

4.6. Реалізація back-end частини

Для реалізації сервер частини була обрана платформа Java та фреймворк - Spring-Boot. Spring Boot - це модуль Spring, спрямований на спрощення використання фреймворку Spring для розробки невеликих програм на Java. Він дозволяє створювати автономні програми на базі Spring, які можна просто запустити. Це забезпечується тим, що Spring Boot має вбудований сервер HTTP, автоматичну конфігурацію і багато іншого, що робить розробку простішою[7].

У проекті був використаний модуль Spring - spring-boot-starter-web. Даний модуль дозволяє дуже легко налаштовувати і розробляти Rest Api для додатків.

Серверна частина розробленого програмного продукту надає такі енд-поінти:

- /api/ontologyclasses
  - get - повертає інформацію про усі класи онтології;
- /api/ontologyclass/info
  - get - повертає детальну інформацію про один клас;
- /api/instances/direct
  - get - повертає усі інстранси одного класу
- /api/instances

- get - повертає усі інстанси;
- put - редагує інстанс;
- post - створює інстанс;
- delete - видаляє інстанс;

- /api/ontologies
  - get - повертає усі онтології в системі;
  - post - заносить у систему нову онтологію;
  - delete - видаляє із системи онтологію;
- /api/exec
  - post - повертає результат запиту до онтології.

Усі операції із для роботи із онтологіями (/api/ontologies) доступні лише для користувачів із роллю “admin”. Операції для модифікації даних (put, post, delete /api/instances) доступні лише для користувачів із роллю “rw_user”. Усі інші енд-поінти можуть використовуватися користувачами із роллю “r_user”.

На рисунку 4.6 зображено структуру (пакети), із яких складається серверний модуль розробленої системи.

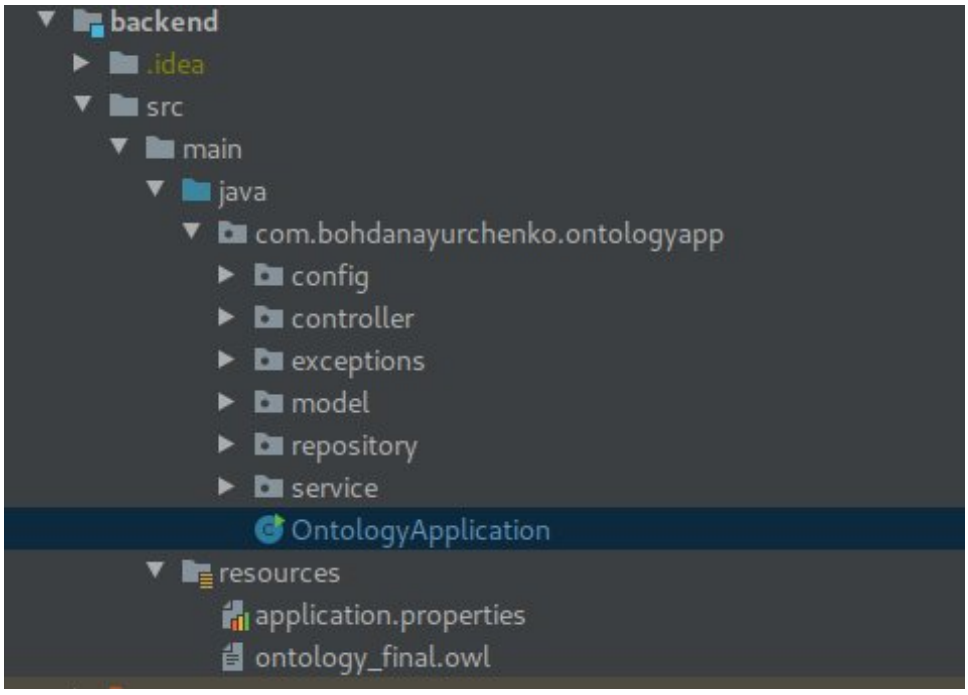


Рисунок 4.6 - Пакети сервер-частини програмного продукту

Сервер частина складається із таких пакетів:

- config - містить класи для конфігурації Spring Boot додатку;
- constroller - містить рест-контроллери, що обробляють запити;
- exceptions - містить класи із виключеннями;
- model - містить класи для представлення даних, якими оперує програма;
- repository - містить класи, що безпосередньо працюють із представленням даних у вигляді онтології;
- service - містить класи, що виконують усю основну роботу.

4.7. Реалізація авторизації та захисту даних

Надання автентифікації та авторизації компонентів програми, які не є загальнодоступними, є важливою частиною кожної системи. Для реалізації цієї чатини програмного продукту був використаний Keycloak.

Keycloak - це рішення з відкритим кодом ідентифікації та управління доступом, яке спрямоване на сучасні програми та послуги. Keycloak дозволяє легко захищати програми та сервіси майже без коду. Він також має багато важливих функцій, що доступні майже із коробки і вимагають лише невеликої конфігурації.

Використання Keycloak для керування доступом у системі означає, що користувачі автентифікуються за допомогою Keycloak, а не самої системи. Після входу в Keycloak користувачам не доведеться знову входити, щоб отримати доступ до іншої частини системи (наприклад до іншого мікросервісу). Таким чином, розроблене програмне забезпечення може бути із легкістю розширене в майбутньому.

Keycloak також реалізовує вхід через соціальні мережі. Дану функціональність теж легко додати через консоль адміністратора. Використання Keycloak також означає, що для додавання такої функціональності до системи не потрібно нічого змінювати в самій програмі[6].

Всі налаштування доступу до системи здійснюються через консоль адміністратора. Адміністратори можуть централізовано керувати всіма аспектами сервера Keycloak. Наприклад, вмикати та вимикати різні функції або керувати користувачами, включаючи сесії, ролі, створювати нових користувачів, змінювати їх дані, видаляти користувачів тощо.

Для захисту програмного продукту була обрана авторизація із використанням OpenID Connect протоколу.

OpenID Connect (OIDC) - протокол аутентифікації, заснований на сімействі специфікацій OAuth 2.0. Він використовує прості веб-токени JSON (JWT)[8]

Таким чином, схема взаємодії різних частин розробленої системи трохи відрізняється від класичної схеми взаємодії модулів SPA (Single Page Application) додатку (рисунок 4.7).

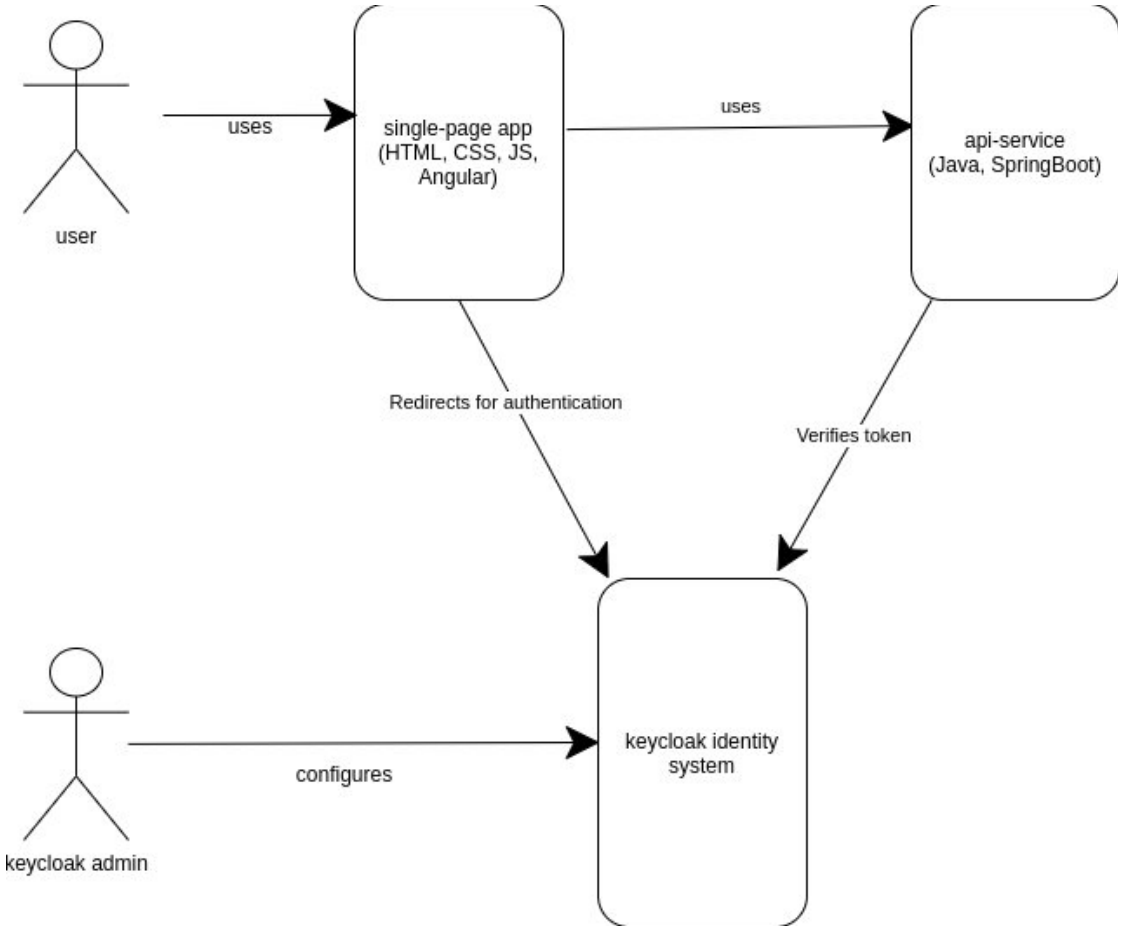


Рисунок 4.7 - Схема взаємодії компонентів SPA із підключеним Keycloack

Для взаємодії сервер частини та системи керуванням доступу був використаний модуль - keycloak-spring-boot-starter. Для взаємодії клієнтської частини із сервером Keycloack був використаний пакет keycloak-angular.

Таким чином у даному розділі було описано програмну реалізацію системи, що розроблялася. Було обґрунтовано вибір технологій та пояснено деякі деталі реалізації. Також було пояснено плюси обраної архітектури системи.

Система вийшла гнучкою і розширюваною, що дозволить їй розвиватися у майбутньому [12].

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для успішної установки, на машині має бути встановленим Docker або podman (аналог Docker для дистрибутивів Linux Fedora і Centos). Якщо на машині буде також Docker Compose - це значно спростить процес розвертання програмного продукту, але задача і без цього програмного забезпечення доволі таки не складна. Для успішної інсталяції достатньо знати основи роботи із ОС Linux і терміналом, уміти встановити потрібні пакети і переходити між директоріями.

На сьогоднішній день Docker також може бути встановлений на Windows. Це означає, що запуск програмного рішення на цій операційній системі теоретично можливий, але тести не проводились.

Машина повинна мати не менше 2 гб оперативної пам'яті а також має бути доступною кінцевим користувачам. Це означає, що машина повинна мати статичну айпі-адресу. В іншому разі, сервіси будуть доступні лише в локальній мережі.

5.1. Установка і налаштування системи

Розроблений продукт - це веб-додаток. Це означає, що він складається мінімум із 2 (у даному випадку 3) окремих сервісів, які треба встановити, запустити і пов'язати між собою. Розроблений програмний продукт складається із 3 окремих модулів. Для кожного із них існує створений образ для Docker.

Для сервісу Keycloak був використаний образ із офіційного сайту - quay.io/keycloak/keycloak. Для API був створений контейнер із назвою ontology-back-end-app на основі офіційного образу openjdk:11, для front-end частини був створений контейнер із назвою ontology-front-end-app на основі nginx:1.15.8-alpine.

Два створених образи не були збережені в Docker Hub, тому для старту сервісів їх потрібно мати локально.

Таким чином. існує декілька сценаріїв інсталяції і розвертання сервісів.

5.1.1. Розвертання усіх сервісів на одній машині за допомогою Docker Compose

Для цього потрібно на машину із деяким дистрибутивом лінукса. Встановлений Docker і Docker Compose останніх версій, налаштовані таким чином, щоб працювати без sudo. Потрібен бути доступ до інтернету.

Порти 8085, 8081, 4200 повинні бути вільними.

Скопіювати директорію із проектом, що поставляється. Після того, як проект був скопійований, потрібно відкрити термінал і перейти в кореневу папку проекту.

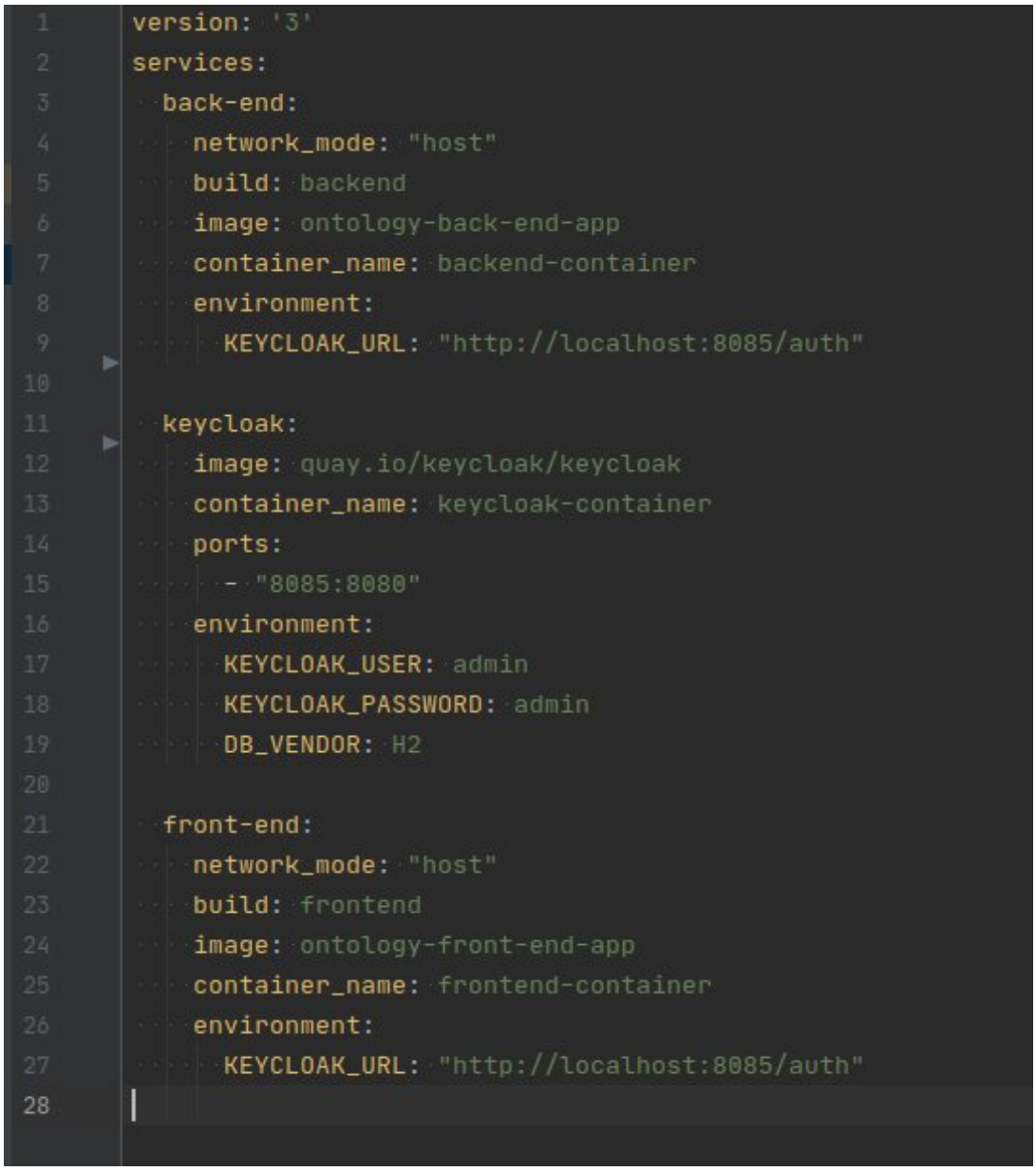
В цій папці потрібно ввести команду “docker-compose up”.

Система сама скомпілює код програмного забезпечення, збере локально образи контейнерів із запустить контейнери із потрібними конфігураціями за замовчуванням.

Якщо усі попередні налаштування системи були зроблені вірно - через деякий час програмний продукт буде доступний на 4200 порту машини. На рисунку 5.1 зображено вміст файлу docker-compose.yml із конфігураціями за замовчуванням, що дозволяють підняти запустити весь проект однією командою.

Конфігурація описує 3 сервіси, що пов'язані між собою. Оскільки сервіс для авторизації і аутентифікації використовується як клієнською так і сервертною частиною - вони від нього залежать. Через те, що сервіс може бути запущений на іншій машині чи хостингу - його адреса була винесена у змінні середовища. Таким чином, у разі зміни адреси сервісу keycloack, єдине. що потрібно буде змінити - це адреси у даному файлі конфігурацій.

Саме таким чином дані інструменти економлять час і кошти.



```yaml
version: '3'
services:
  back-end:
    network_mode: "host"
    build: backend
    image: ontology-back-end-app
    container_name: backend-container
    environment:
      KEYCLOAK_URL: "http://localhost:8085/auth"

  keycloak:
    image: quay.io/keycloak/keycloak
    container_name: keycloak-container
    ports:
      - "8085:8080"
    environment:
      KEYCLOAK_USER: admin
      KEYCLOAK_PASSWORD: admin
      DB_VENDOR: H2

  front-end:
    network_mode: "host"
    build: frontend
    image: ontology-front-end-app
    container_name: frontend-container
    environment:
      KEYCLOAK_URL: "http://localhost:8085/auth"
```

Рисунок 5.1. - Конфігураційний файл docker-compose із конфігураціями за замовчуванням

5.1.2. Розвертання усіх сервісів на одній машині без Docker Compose

Якщо за якихось причин Docker Compose не вдалося встановити чи налаштувати - проект можна запустити самотужки декількома командами. Для цього потрібно на машину із деяким дистрибутивом Linux . Встановлений Docker або Podman останньої версії, налаштовані таким чином, щоб працювати без sudo. Потрібен бути доступ до інтернету. Порти 8085, 8081, 4200 повинні бути вільними.

Спочатку потрібно скопіювати директорію із проектом, що поставляється. Після того, як проект був скопійований, потрібно відкрити термінал і перейти в кореневу директорію проекту.

В цій директорії потрібно ввести команду “docker build -t ontology-back-end-app ./backend/”.

У разі успішного виконання в системі з’явиться образ із назвою ontology-back-end-app. Перевірити наявність образу можна за допомогою команди “docker images | grep ontology-back-end-app“. Потрібний результат виконання команди зображений на рисунку 5.2.

Рисунок 5.2 - Результат виконання команди

У разі успіху треба виконати наступну команду “docker build -t ontology-front-end-app ./frontend/”.

У разі успішного виконання в системі з’явиться образ із назвою ontology-front-end-app. Перевірити наявність образу можна за допомогою команди “docker images | grep ontology-front-end-app“. Приклад результату виконання команди зображений на рисунку 5.3.

Рисунок 5.3 - Результат виконання команди

Після успішного виконання попередніх команд можна приступити до створення контейнерів. Почати варто із контейнера із сервісом Keycloack. Для цього в командному рядку треба ввести таку команду: “docker run -d -p 8085:8080 -e KEYCLOAK_USER=admin -e KEYCLOAK_PASSWORD=admin --name keycloack_container quay.io/keycloak/keycloak”.

У разі успіху створиться контейнер із назвою keycloack_container. Перевірити успішність можна за допомогою команди “docker ps”. Приклад результату виконання команди зображений на рисунку 5.4.

```
[bohdanayurchecko@lee o-m-s]$ docker ps -a
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS    PORTS                                   NAMES
781ce635c695   quay.io/keycloak/keycloak  "/opt/jboss/tools/do…"   21 seconds ago   Created   8443/tcp, 0.0.0.0:8085->8080/tcp   keycloack_container
```

Рисунок 5.4 - Результат виконання команди

Вона створить і запустить контейнер із назвою “keycloack_container”, використовуючи останню версію образу Keycloak із репозиторію образів. Також вона прив'яже порт 8085 фізичної машини до 8080 порту контейнеру (порт, що використовує Keycloak). Також ця команда створить користувача адміністратора “admin” із паролем “admin” - це логін і пароль до панелі налаштувань серверу.

Під час старту контейнеру можна вказати свої логін і пароль для адміністратору панелі налаштувань. Також можна вказати свій порт (його варто запам'ятати, адже він важливий для інших контейнерів).

Якщо контейнер буде запущений в якомусь хостингу для контейнерів - дана команда може прийняти вигляд конфігурації. Далі наданий приклад конфігурації, що була використана для запуску сервера на платформі sloppy.io (рисунок 5.5).

У якості бази даних в конфігураціях вказана in-memory база даних - H2. Це означає, що після перезапуску контейнера усі налаштування пропадуть. Тому для роботи із програмою варто налаштувати використання окремої бази даних, наприклад postgresql. Інструкції, як це зробити, можна знайти в документації на офіційному сайті Keycloak .

Після успішного запуску контейнеру із Keycloack сервером потрібно підкючитись до панелі адміністратора. Для цього треба набрати у браузері адсерсу сервісу. Для того, аби авторизуватися, потрібно вказати логін і пароль, що були вказані у команді старту контейнеру.

Після успішної авторизації потрібно імпортувати файл із налаштуваннями реалму для розробленої системи. Файл входить в поставку розробленого продукту. Файл має усі базові налаштування, такі як реалм, система, ролі користувачів.

Безпосередньо користувачів у налаштуваннях немає, їх треба створити вручну. Детальні інструкції щодо створення користувачів і інших налаштувань можна знайти на офіційному сайті розробки та в офіційній документації. Також , оскільки продукт є популярним і безперечно використовується у багатьох системах, існує велика кількість майстер-класів і уроків.

Рисунок 5.5 - Приклад конфігурації сервісу для хостингу sloppy.io

Після успішного налаштування серверу потрібно запустити контейнер із сервер частиною програмного продукту. Це можна зробити за допомогою команди : “docker run -d -e KEYCLOAK_URL=http://localhost:8085/auth --name backend_container ontology-back-end-app”

Після цього потрібно запустити контейнер із front-end частиною програмного продукту. Це можна зробити за допомогою команди “docker run -d -e

KEYCLOAK_URL=http://localhost:8085/auth --name frontend-container ontology-front-end-app”

У разі, якщо сервіс із keycloack знаходиться не за адресою http://localhost:8085/auth (за замовчуванням), потрібно вказати його справжню адресу.

В кінці потрібно перевірити успішність усіх операцій за допомогою команди : docker ps”.

У списку повинно бути 2 - 3 контейнери (в залежності від того, як змінилися налаштування).

Після цього можна приступати до подальших конфігурацій (створення користувачів тощо), що описані у попередньому розділі.

5.1.3. Розвертання сервісів на окремих машинах

У разі, якщо сервіс keycloak був запущений на іншій машині, або на окремому хостингу, конфігурація docker-compose зміниться. На рисунку 5.6 зображено конфігурацію сервісів у випадку, якщо сервіз із keycloack був запущений на іншій машині чи хостингу і має url keycloackurl .

```yaml
version: '3'
services:
  back-end:
    network_mode: "host"
    build: backend
    image: ontology-back-end-app
    container_name: backend-container
    environment:
      KEYCLOAK_URL: "http://keycloackurl/auth"

  front-end:
    network_mode: "host"
    build: frontend
    image: ontology-front-end-app
    container_name: frontend-container
    environment:
      KEYCLOAK_URL: "http://keycloackurl/auth"

```

Рисунок 5.6 - Приклад конфігурації сервісів

5.2. Методика роботи із програмним продуктом

Для користувачів із різними ролями початок роботи із системою починається у різний час і різним чином.

5.2.1. Початок роботи для адміністратора kaycloak

Початок роботи із розробленою системою для адміністратора Keycloak починається під час запуску контейнера із даним модулем, коли вказуються логін і пароль для адміністратора.

Після успішного запуску системи адміністратор має створити реалм для розробленої системи, налаштувати клієнта для реалма (рисунок 5.9) (розроблений веб-додаток), створити ролі користувачів (рисунок 5.7) (усі потрібні конфігурації для даних налаштувань містяться у файлі конфігурацій, що поставляється як частина розробленої системи). Для налаштування системи із допомогою файлу із конфігураціями потрібно натиснути кнопку для створення нового реалму і замість створення із нуля - обрати опцію імпорту файлу із конфігураціями.

І нарешті після цього створити перших користувачів (рисунок 5.8) (користувачів немає у файлі конфігурації), що зможуть працювати безпосередньо із розробленою онтологічною системою для аналізу міжнародної діяльності. На рисунках зображені приклади конфігурації додатку, сторінка конфігурації ролей у додатку і створені користувачі, що мають доступ до розробленого веб-додатку.

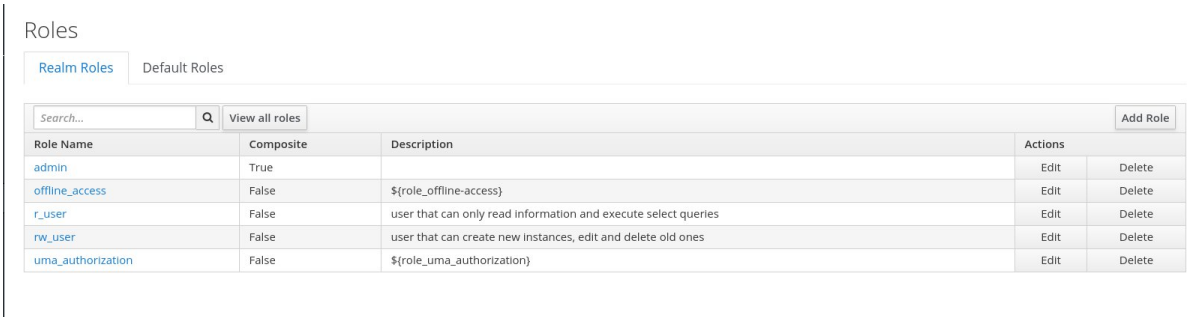


Рисунок 5.7 - Сторінка менеджменту ролей

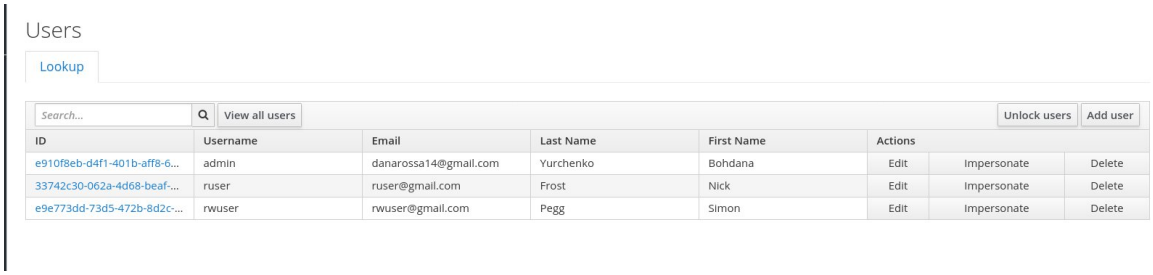


Рисунок 5.8 - Сторінка менеджменту користувачів системи

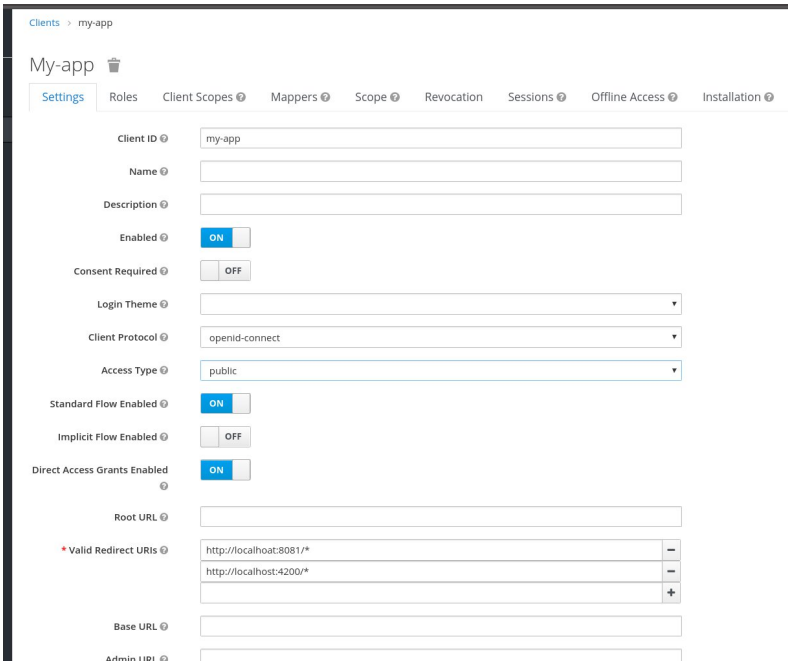


Рисунок 5.9 - Конфігурації підключеного додатку у Keycloack

5.2.2. Початок роботи для користувача із роллю “admin”

Початок роботи із системою для користувача із роллю “адміністратор” починається після створення адміністратором keycloack такого користувача із роллю “адміністратор” і видачею логіну і паролю для входу до системи.

Вперше зайшовши у систему, іще не авторизований користувач опиняється на головній сторінці, що описує головні функції розробленої системи (рисунок 5.10).

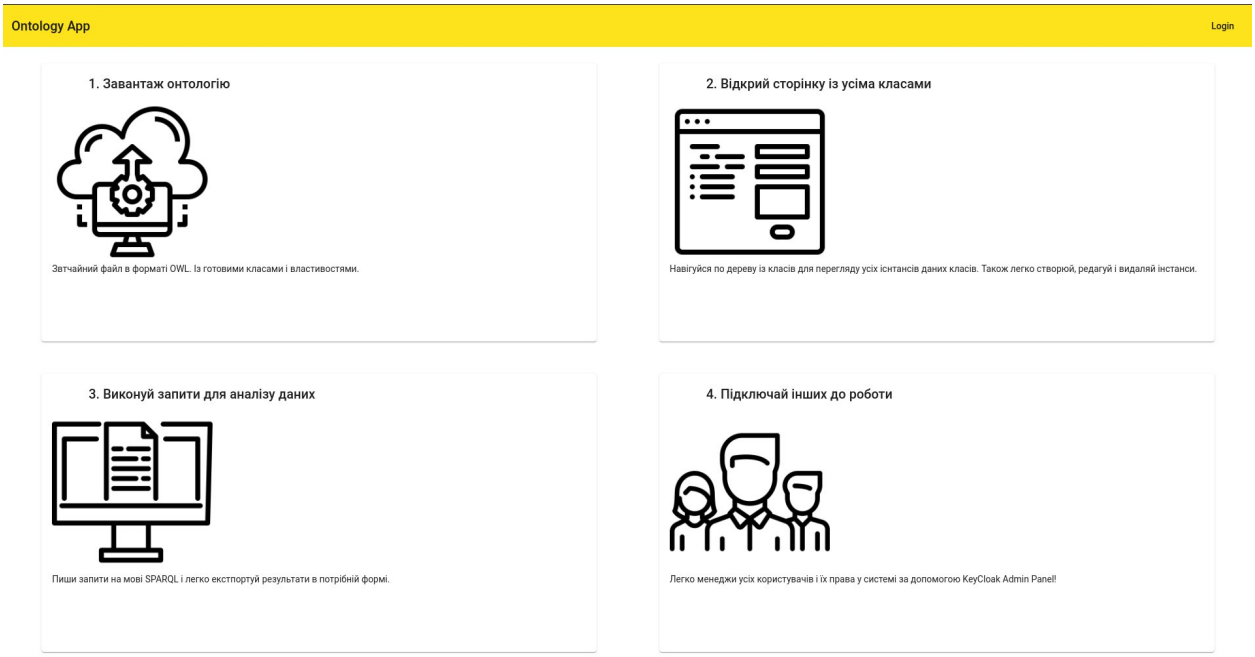


Рисунок 5.10 - Головна сторінка для незареєстрованого користувача

Сторінка має лише одну активну кнопку - кнопку для логіну. Після її натискання користувач перенаправляється на сторінку Keycloack для введення логіну і паролю (рисунок 5.11), що йому були видані адміністратором системи авторизації.

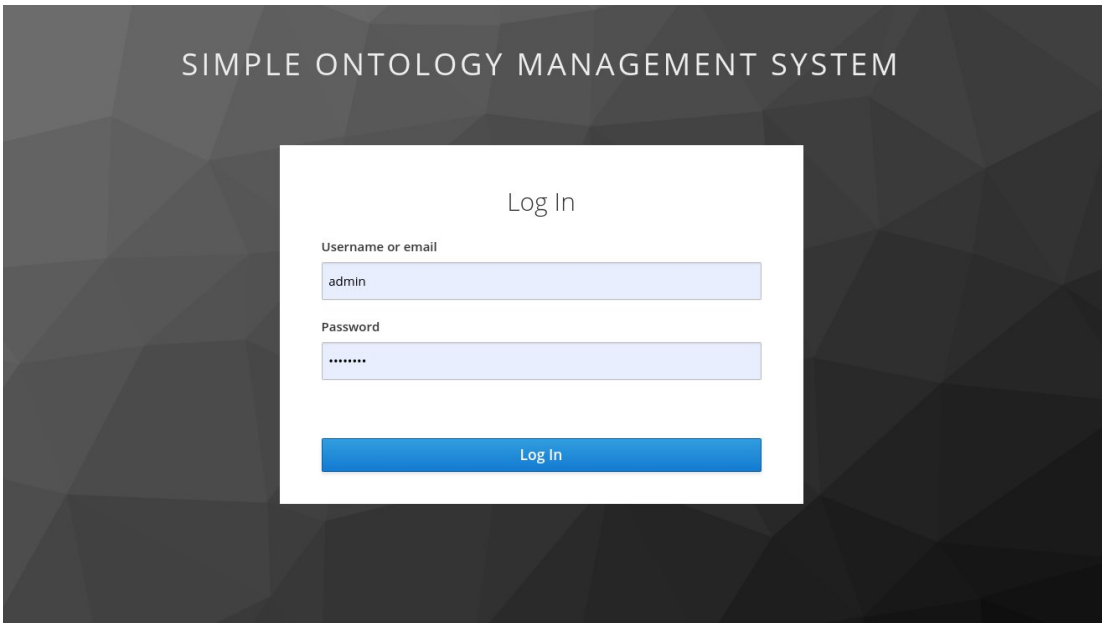


Рисунок 5.11 - Сторінка авторизації

Після того, як користувач правильно ввів логін і пароль, він може приступити до роботи у системі.

Після авторизації головна сторінка має не лише опис головних функцій, а й посилання на сторінки, на яких ці функції є доступними (рисунок 5.12).

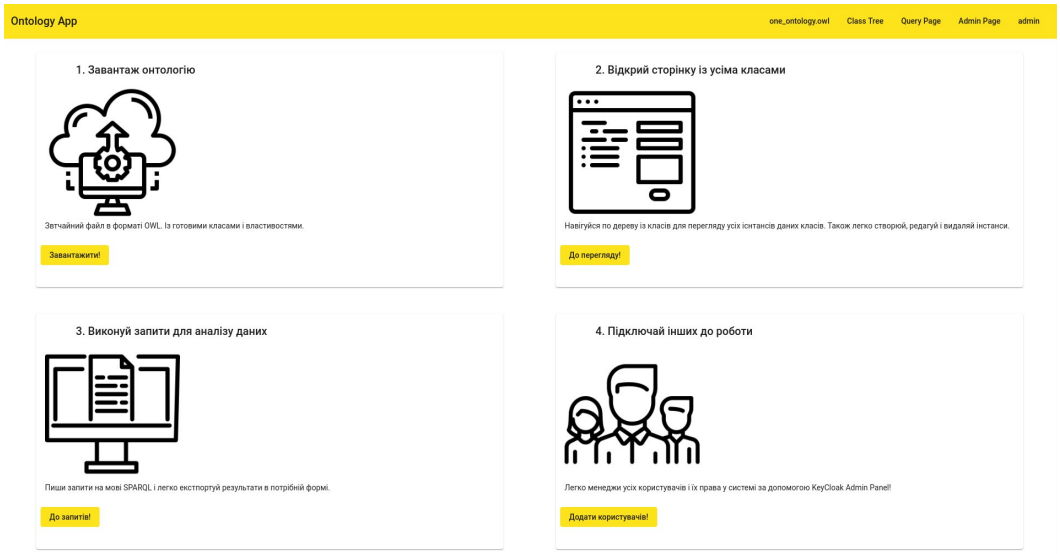


Рисунок 5.12 - Головна сторінка для користувача із роллю “admin”

Перш ніж роботу із системою зможуть почати користувачі із ролями “r_user” та “rw_user” - адміністратор має імпортувати у систему онтології, із якими інші користувачі зможуть працювати (вводити дані та переглядати їх). Оскільки система задля простоти використання була реалізована лише із основними функціями для редагування інстансів у онтології - у систему має бути імпортована онтологія із готовими класами і їх властивостями.

Це може зробити лише адміністратор на сторінці із назвою “Admin page” (рисунок 5.13).

На цій сторінці є таблиця із усіма онтологіями у системі і їх версіями, а також форма для завантаження нових онтологій або версій. Кожен файл, що є у таблиці, можна скачати на локальну машину, відредагувати (наприклад, додати класи, використовуючи професійне середовище розробки - Protege) і завантажити назад.

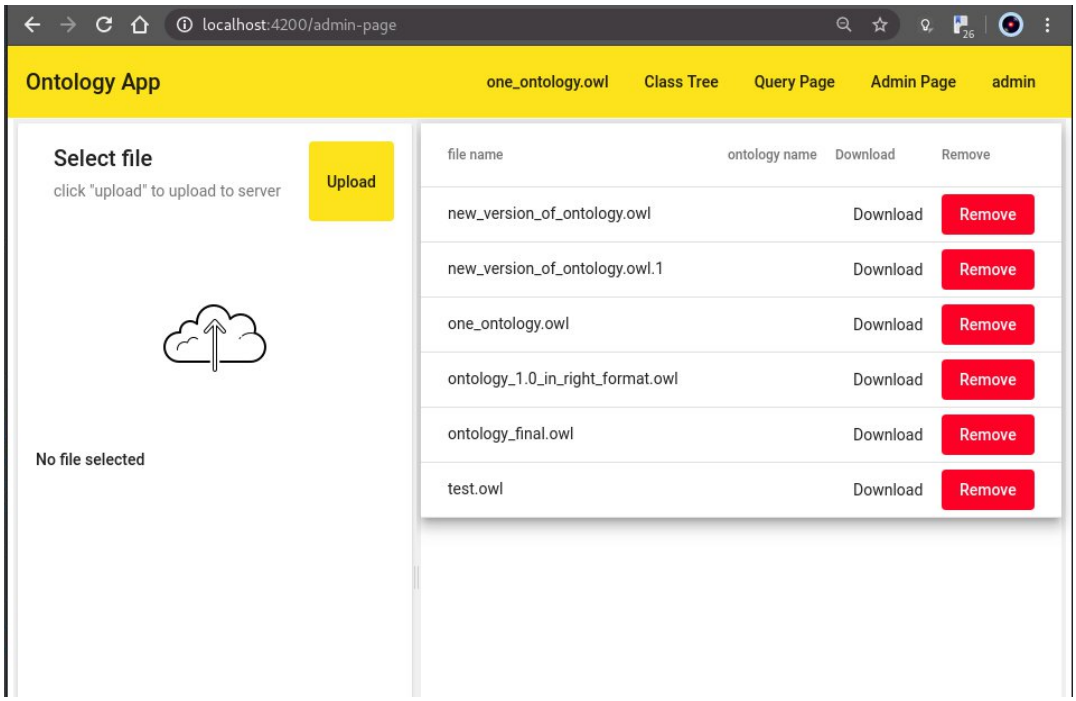


Рисунок 5.13 - Сторінка менеджменту онтологій

Якщо у систему завантажують файл, що уже є в системі, то попередній файл зберігається із індексом-номером версії (1, 2, 3). А новий файл зберігається без індексу на кінці - це означає, що це актуальна версія онтології (рисунок 5.14). Таким чином, у системі реалізована функція контролю версій. Якщо додані адміністратором класи або властивості класів виявились неправильні, завжди можна відкотитись до попередньої версії онтології.

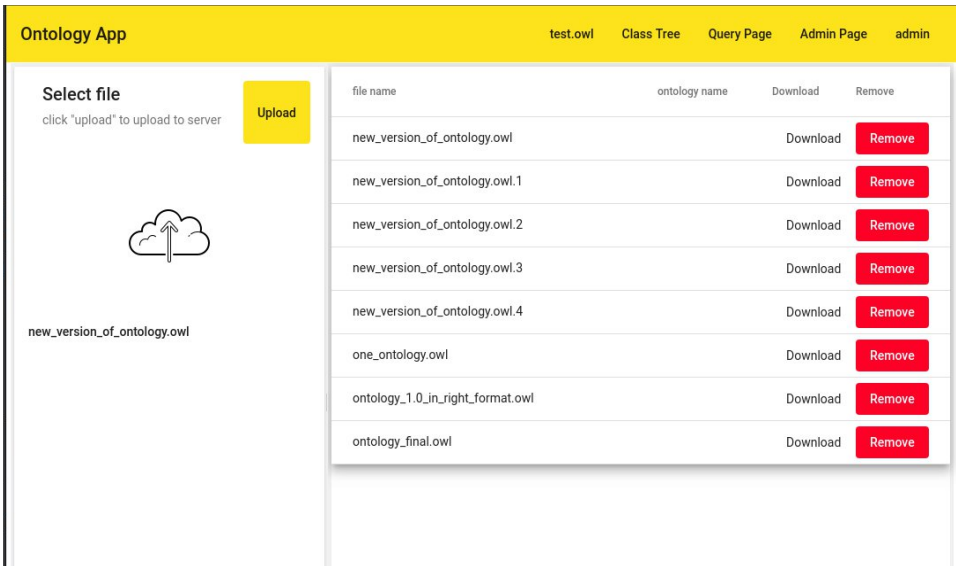


Рисунок 5.14 - Багато версій однієї онтології

Непотрібні онтології або старі версії онтологій можуть бути видалені із системи.

Основним інструментом для навігації у системі є шапка сайту. Тут є такі елементи (зліва направо) (рисунок 5.15):

- назва додатку (кнопка для переходу на головну сторінку)
- кнопка для вибору поточної онтології (відображає назву поточної онтології)
- посилання на сторінку із деревом класів для перегляду інстансів у вигляді таблиць, а також для редагування інстансів
- посилання на сторінку для менеджменту онтологій
- кнопка поточного користувача.

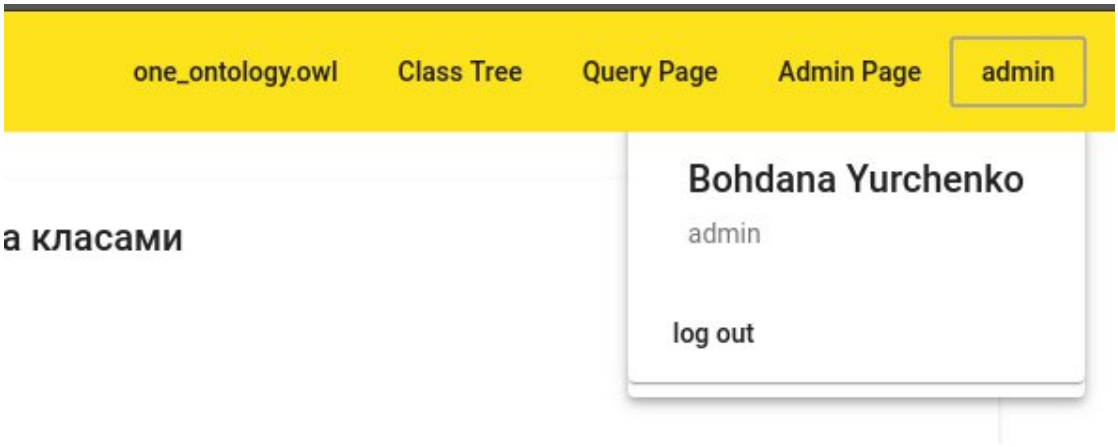


Рисунок 5.15 - Кнопки головної панелі веб-додатку

Кнопка поточного користувача завжди показує логін користувача, що працює у системі. При кліку по ній з'являється меню, що показує ім'я та прізвище, а також роль поточного користувача, а також кнопку для виходу із системи .

На цьому необхідні дії адміністратора закінчуються та інші користувачі можуть приступити до роботи.

5.2.3. Початок роботи для користувача “rw_user”

Після того, як адміністратор системи завантажив у систему хоча б одну онтологію із класами і їх властивостями, користувачі із роллю “rw_user” можуть почати свою роботу - вносити інформацію у систему.

Після авторизації, яка проходить так само, як і для користувача-адміністратора, користувач має обрати онтологію, із якою він хоче працювати (наприклад передивлятись або вносити дані).

При першому вході у систему ніяка онтологія не буде обрана. На панелі (шапці сайту) буде напис про те, що ніяка онтологія не обрана. Для того, аби обрати онтологію, треба натиснути на даний напис. Тоді з'явиться діалог із полем вибору онтології із усіх, що є у системі (рисунок 5.16).

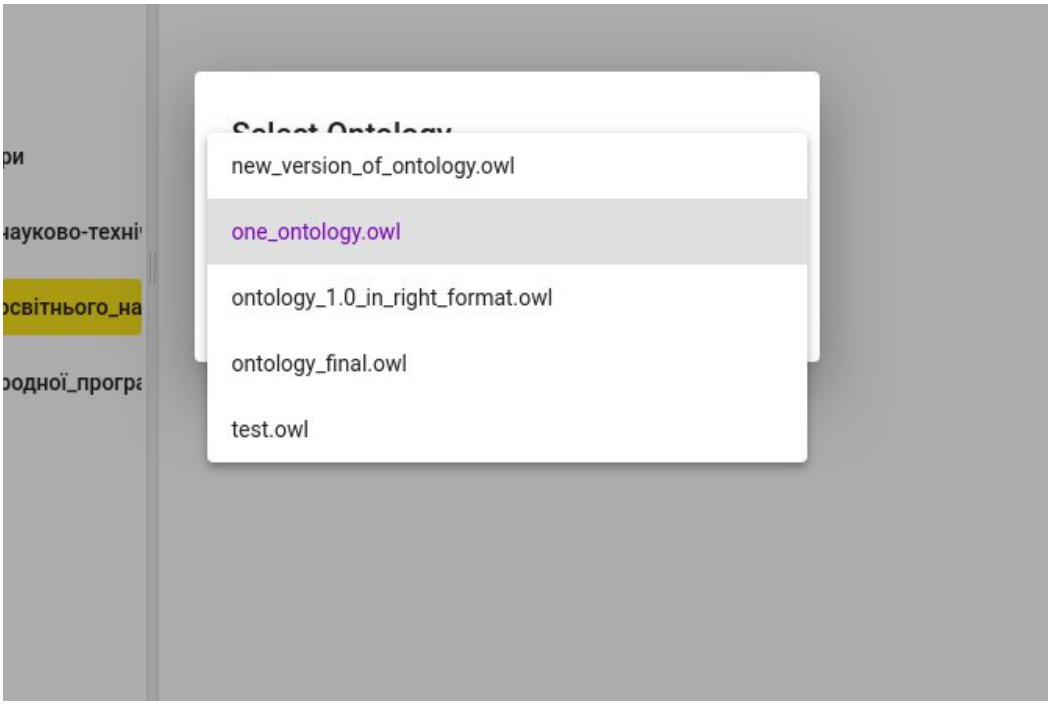


Рисунок 5.16 - Вибір активної онтології для користувача

У даному списку містяться лише актуальні версії онтологій. Це означає, що, наприклад, якщо онтологія із назвою “test.owl” була декілька разів завантажена адміністратором і у списку усіх файлів вона з'являється декілька разів із номерами

1,2,3, і так далі, то у даному списку буде лише остання версія, що не має порядкового номеру.

Після вибору онтології треба натиснути кнопку “Select” (рисунок 5.17) - і обрана онтологія запишеться у налаштування користувача (рисунок 5.18). Дане налаштування зберігається у браузері користувача, тому, наступного разу, коли користувач ввійде у систему із того ж пристрою, значення обраної онтології збережеться і її не прийдеться обирати повторно.

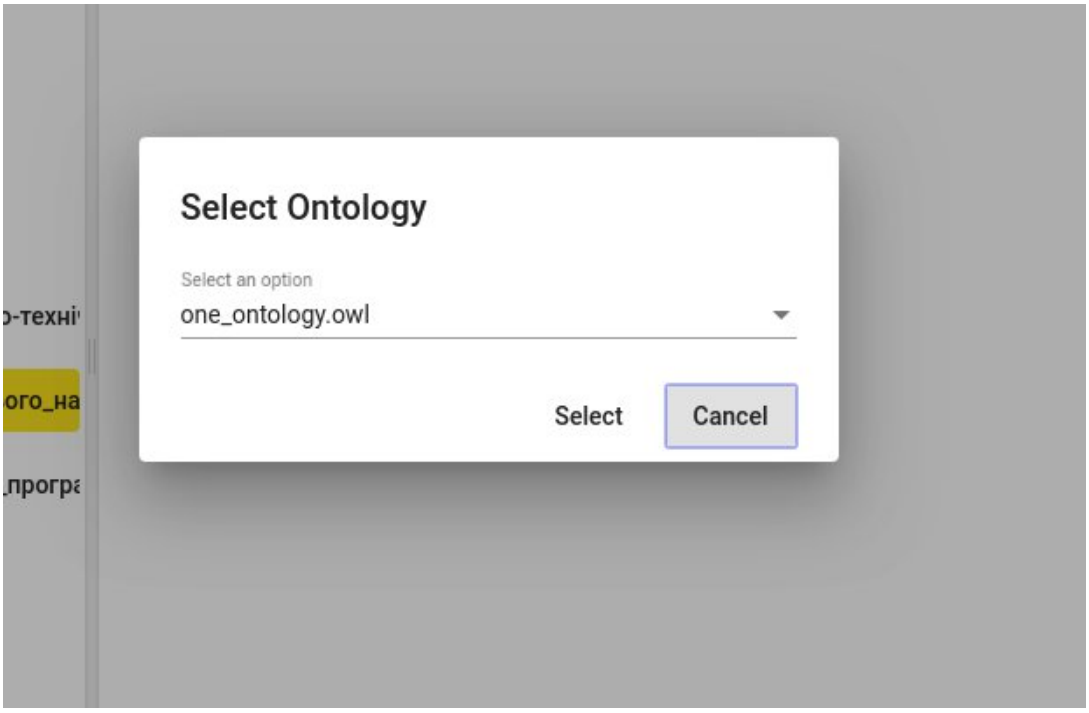


Рисунок 5.17 - Обрана активна онтологія

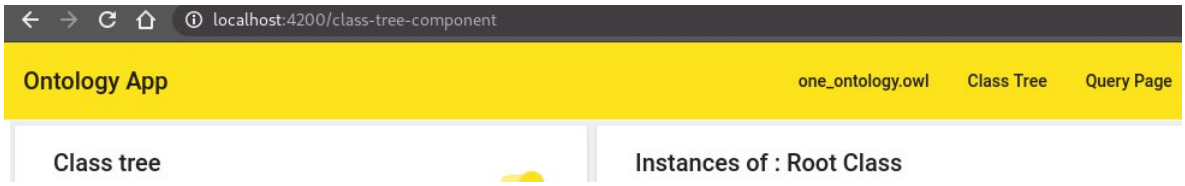


Рисунок 5.18 - Установлена активна онтологія

Після того, як була вказана онтологія, з якою хоче працювати користувач, - він може приступити до роботи - занесенню даних.

Даний функціонал знаходиться на сторінці “Class Tree” - сторінці із деревом класів. На цю сторінку можна перейти, натиснувши кнопку “Class Tree” на головній панелі сайту.

Дана сторінка складається із двох частин : панелі із деревом класів онтології і панелі для відображення інстансів онтології (рисунок 5.19).

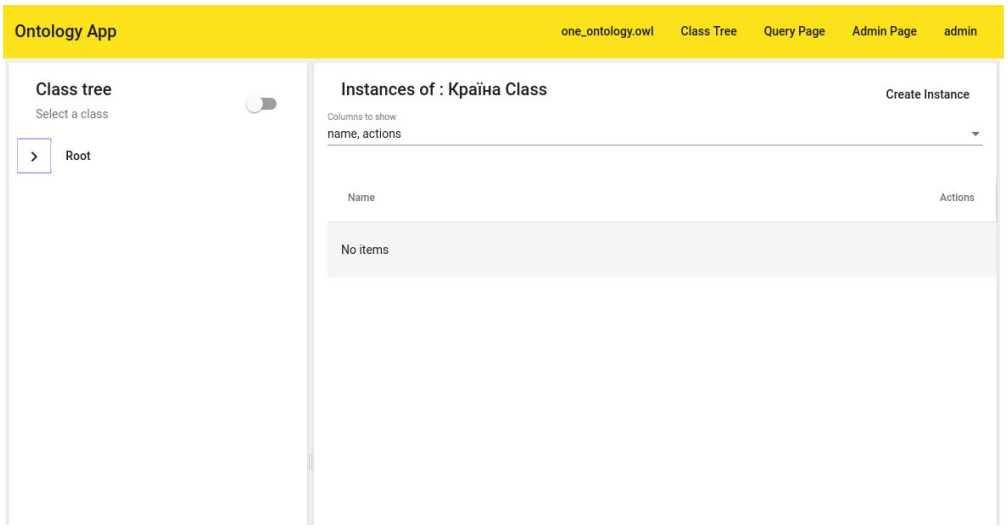


Рисунок 5.19 - Сторінка для із деревом класів онтології

Кореневим класом кожної онтології є “Root class”. Якщо у класу є підкласи, біля його назви є стрілочка-кнопка, за допомогою якої можна розгорнути клас для перегляду його підкласів. Таким чином, можна переглянути всі класи, що є в обраній онтології (рисунок 5.20).

На кожен клас можна натиснути - і він підсвітиться кольором. Це означає, що клас був обраний і права панель сторінки зміниться для перегляду інстансів, що належать обраному класу.

Наприклад, якщо користувач деякий час попрацював і створив декілька інстансів класу “Факультет”, то сторінка буде виглядати так, як зображено на рисунку 5.21.

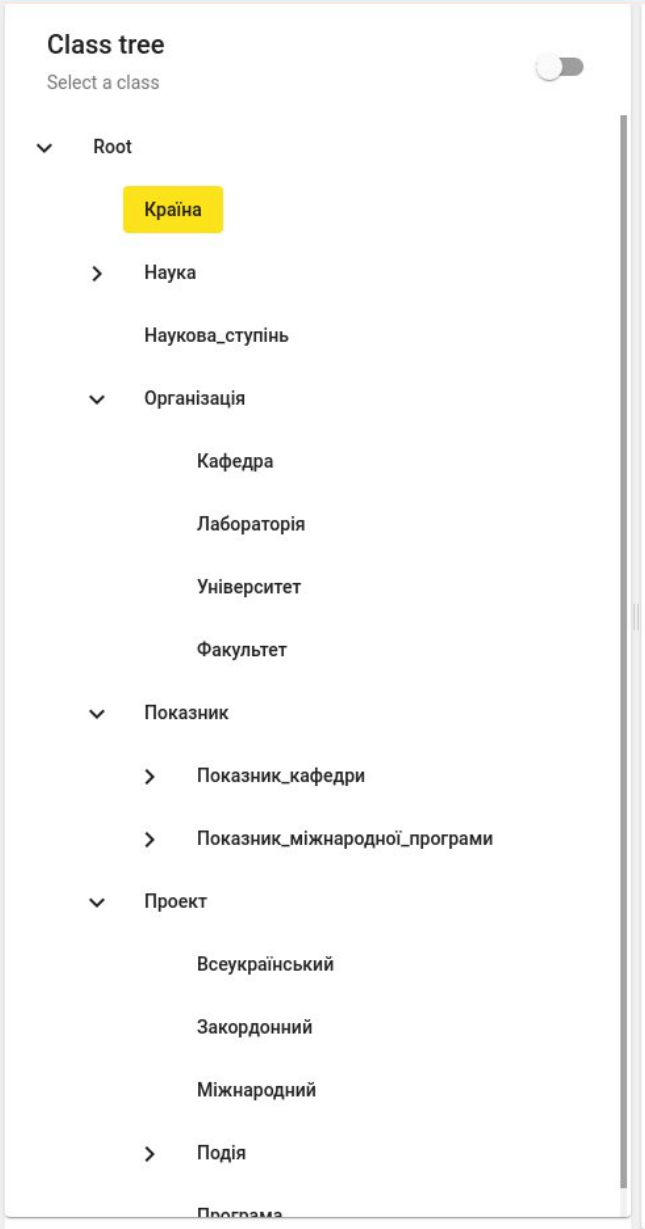


Рисунок 5.20 - Перегляд дерева класів активної онтології

Таблиця із інстансами за замовчуванням показує лише 2 стовпчики, що є стандартними для кожного класу і інстансу - назву, що є частиною URI, та стовпчик із доступними діями.

Якщо ж користувач захоче переглянути усі інстанси класу “Організація ”, то таблиця буде пустою (рисунок 5.22) (незважаючи на те, що клас “Факультет” є підкласом “Організації” і відповідно кожен факультет є організацією). Таблиця буде пустою тому, що за замовчуванням таблиця показує лише інстанси безпосередньо

обраного класу, і в даній онтології не було створено ніякого інстансу класу “Організація”.

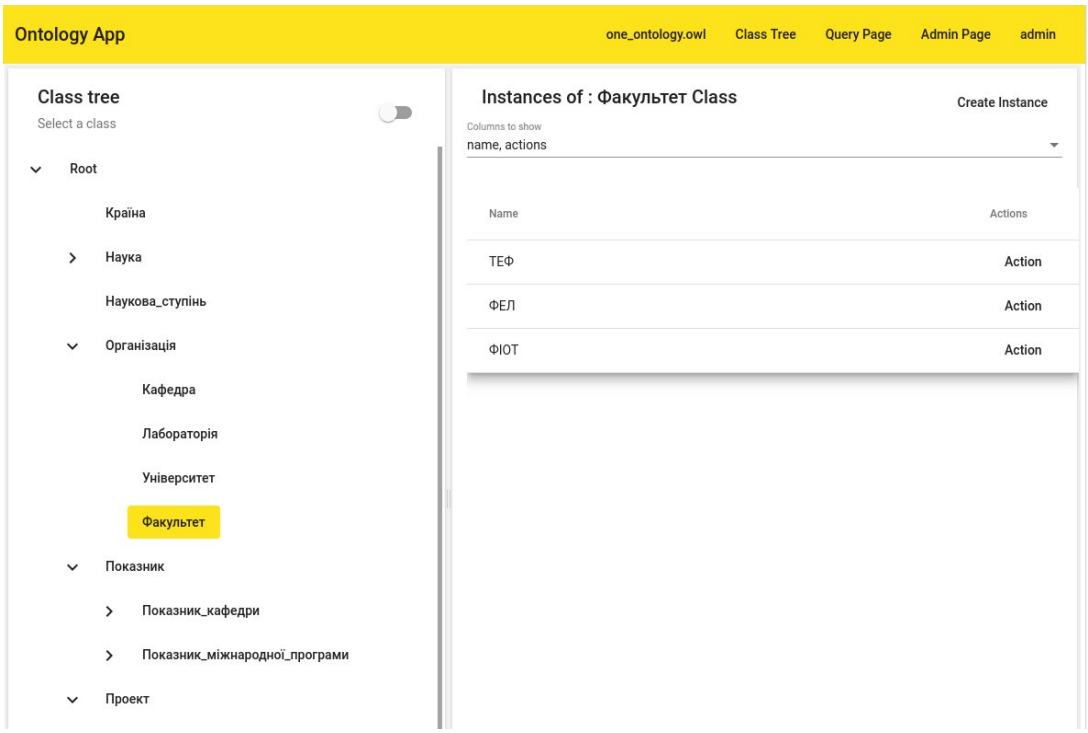


Рисунок 5.21 - Перегляд інстансів обраного класу

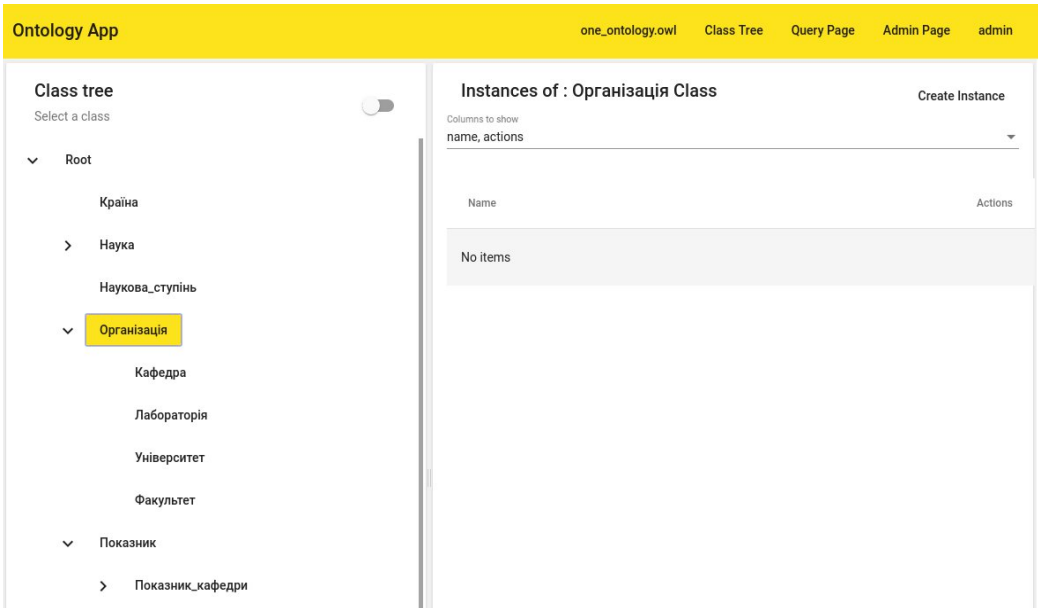


Рисунок 5.22 - Перегляд інстансів батьківського класу

Для того, аби перемкнуть режим перегляду на інший, треба змінити значення кнопки зверху на панелі із деревом класів. Якщо перемикач ввімкнений - режим перегляду встановлений на той, що дозволяє переглядати інстанси обраного класу так само, як і інстанси його підкласів. Наприклад, якщо перемикач ввімкнений і обраний клас “Організація” - користувач побачить усі додані факультети так само, як і університети, кафедри та лабораторії (рисунок 5.23). У даному випадку був створений лише один університет - КПІ.

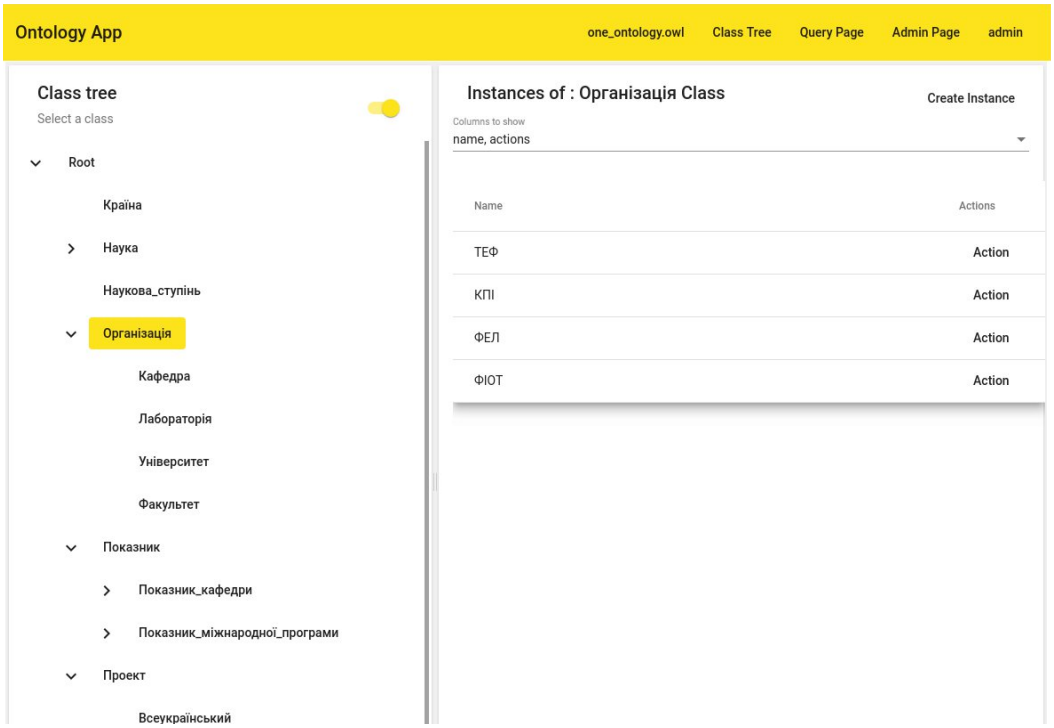


Рисунок 5.23 - Перегляд інстансів обраного батьківського класу із дочірніми класами

За замовчуванням таблиця має лише 2 стовпчики - назву інстансу, що є унікальним ідентифікатором в онтології і доступні дії.

Для кожного класу також є набір унікальних властивостей, що доступні лише для нього. Наприклад, у класу “Організація” є властивість “Назва”, де можна вказати повну назву організації. Даний стовпчик можна додати у таблицю за допомогою поля, що знаходиться над таблицею (рисунок 5.24) .

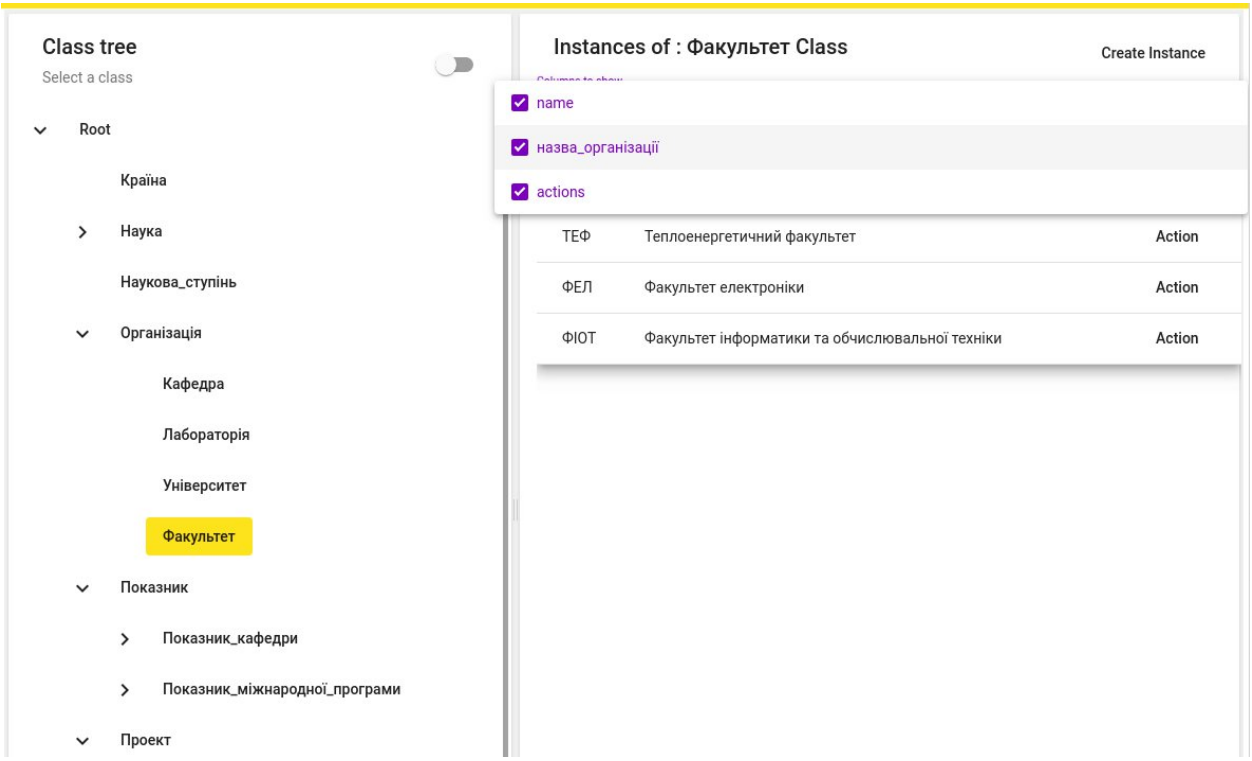


Рисунок 5.24 - Відображення додаткових стовпців у для таблиць інстансів

На скріншоті видно, що як тільки була поставлена галочка біля назви властивості “назва_організації” - даний стовпчик із указаними значеннями для кожного інстансу з’явився у таблиці. Список доступних додаткових стовпчиків унікальний для кожного класу. Таким чином, у класу “Показник” є інші властивості, яких немає у класу “Організація” і які можна відобразити лише, якщо у дереві класів є обраним або клас “Показник”, або якийсь із його дочірніх класів.

На даній сторінці також можна відобразити усі інстанси в онтології незалежно від їх класів. Для цього треба обрати клас ”Root” (що є батьківським для усіх класів в онтології) і ввімкнути опцію показу інстансів усіх дочірніх класів (рисунок 5.25).

Коли на панелі класів є обраний якийсь клас - на панелі із таблицею інстансів з'являється кнопка для створення нового інстансу обраного класу. Ця кнопка доступна лише для ролей “admin” та “rw_user” і лише якщо обраний не кореневий клас (інстанс, якого створити неможливо).

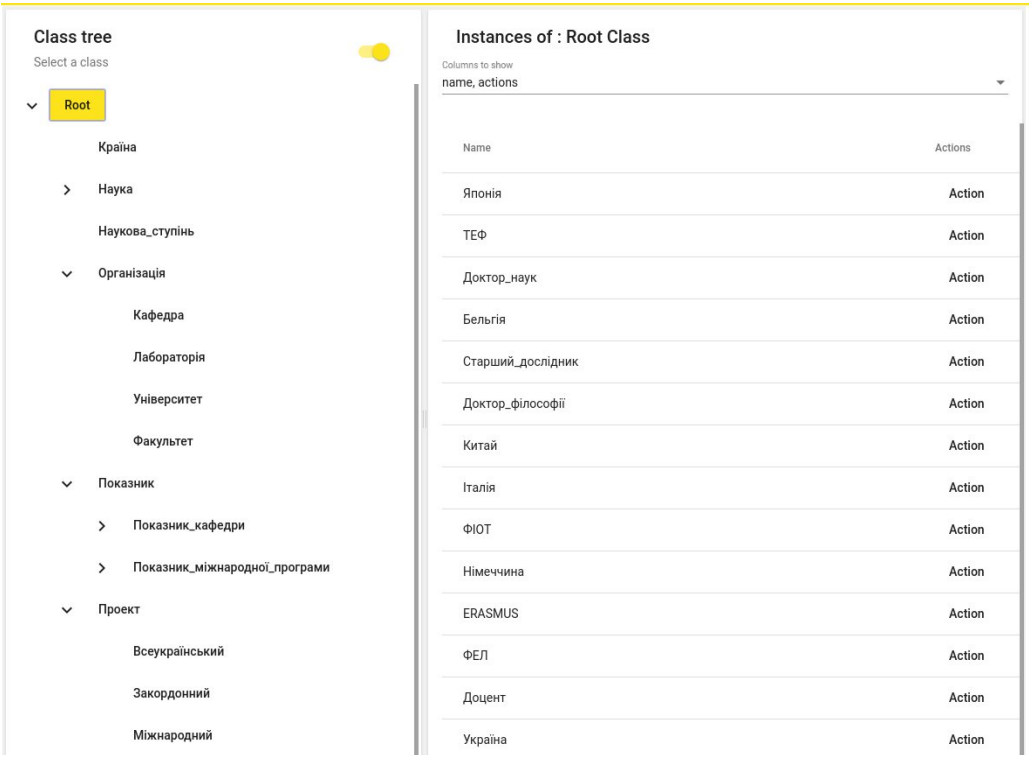


Рисунок 5.25 - Відображення усіх інстансів у онтології

Після кліку по цій кнопці з'являється діалог для створення інстансу із усіма потрібними полями (рисунок 5.26).

Діалог складається із одного головного поля - назви інстансу та полів для усіх властивостей, що є у класу, інстанс якого користувач намагається створити.

Наприклад, клас “Науковий_ступінь” не має ніяких додаткових властивостей. Тому для створення інстансу потрібно лише вказати назву , що є частиною URI інстансу. Через це на назву інстансу накладаються такі обмеження :

- назва має бути унікальною в онтології;
- назва не може містити пробіли;
- назва може складатись із букв.

У разі, якщо вказані значення не є валідними - кнопка збереження не активна (рисунок 5.26).

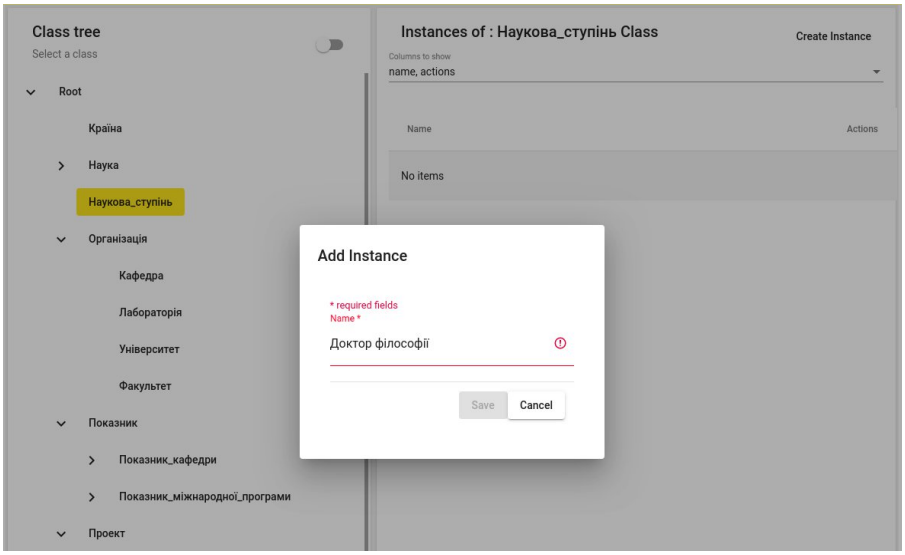


Рисунок 5.26 - Невалідна назва інстансу

Для активації кнопки назва інстансу була змінена таким чином (пробіл був замінений на нижнє підкреслення) (рисунок 5.27).

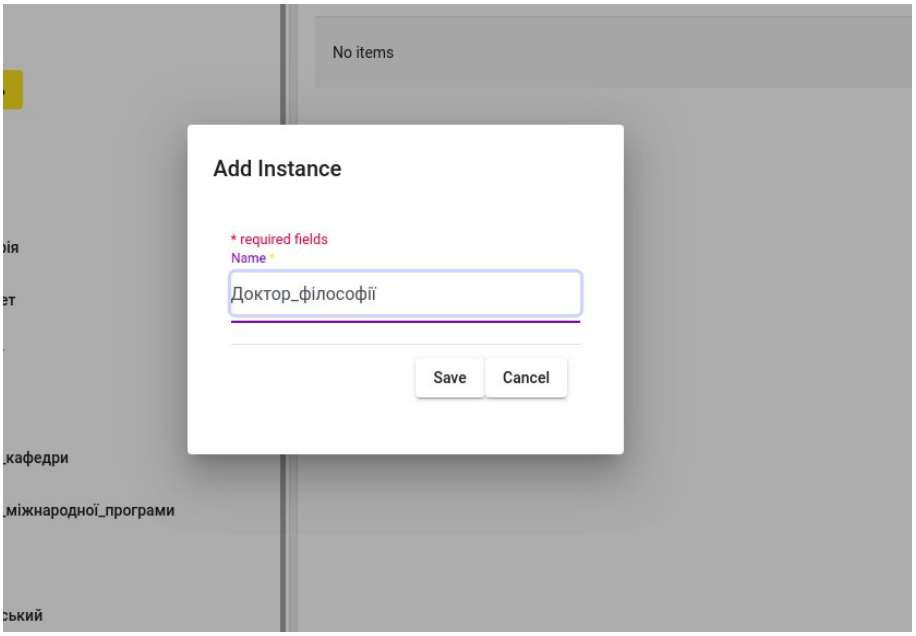


Рисунок 5.27 - Валідна назва інстансу

Якщо клас має додаткові властивості, поля для них з’являться у формі створення інстансу. Наприклад, клас показник кафедри має такі властивості : посилання на кафедру, значення (число), коментар (вільний текст), рік (дата). Якщо властивість є типу “посилання”, то поле дозволить користувачеві обрати інстанс із

можливих. На даному скріншоті видно, кафедру можна вибрати серед усіх кафедр, що є в онтології (для прикладу у систему була додана лише одна кафедра - АПЕПС) (рисунок 5.28).

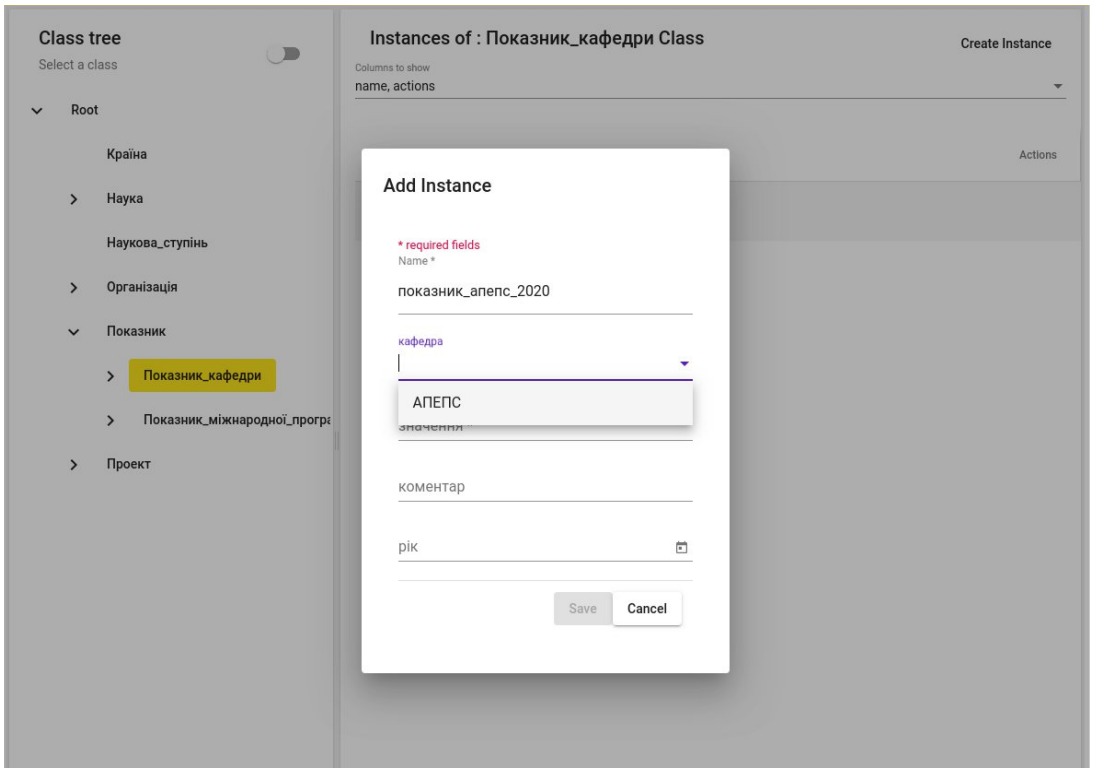


Рисунок 5.28 - Вказування значення властивості типу посилання

Усі створені інстанси можна редагувати або видаляти (ці дії теж доступні лише для користувачів із ролями “admin” та “rw_user”) (рисунок 5.29).

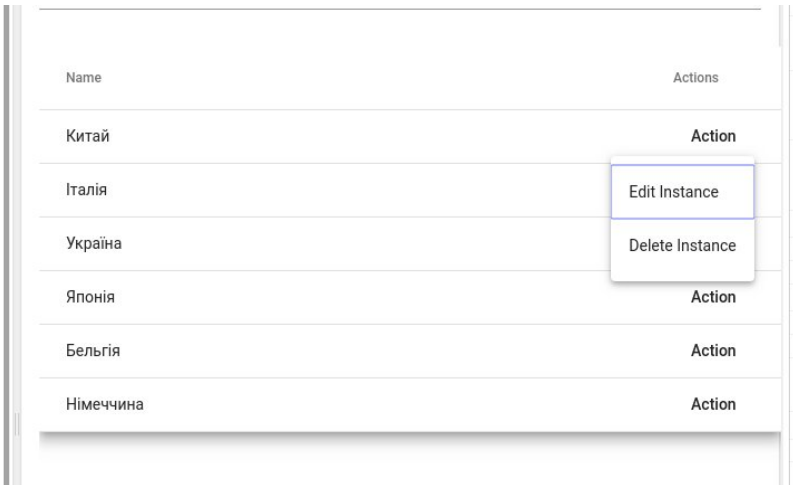


Рисунок 5.29 - Доступні дії

При чому, якщо у дереві класів обраний кореневий клас, то інстанси можна лише видаляти (опція редагування доступна тільки тоді, коли обраний більш конкретний клас).

Після того, як користувачами із правами редагування даних дані були внесені в онтологію, користувачі, що мають права лише для перегляду даних, можуть приступати до роботи.

5.2.4. Початок роботи для користувача “r_user”

Початок роботи користувача із правами читання починається так само, як і у rw_user-а. Користувач має отримати логін і пароль від адміністратора системи і ввійти у систему. Після авторизації користувач може переглядати дані у формі таблиць, користуючись сторінкою із деревом класів.

Але він також може користуватись сторінкою для виконання запитів на мові SPARQL. Для того, аби перейти на дану сторінку, треба натиснути кнопку “Query Page” на головній панелі додатка. Перед написанням першого запиту необхідно обрати активну онтологію, до якої запит буде виконуватись.

Після вибору онтології користувач може написати якийсь запит на мові SPARQL і натиснути кнопку “Run” , що приведе до виконання запиту (рисунок 5.30). На скріншоті для прикладу був написаний запит, що виводить список усіх класів у системі.

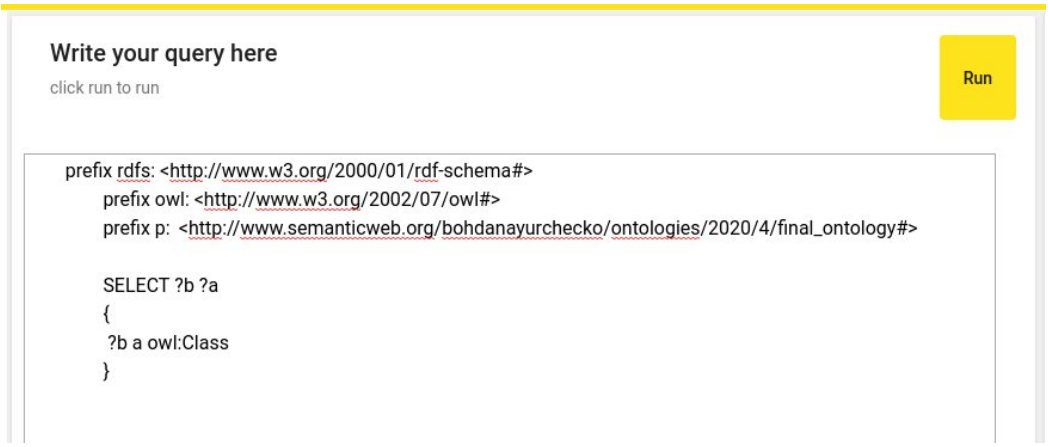


Рисунок 5.30 - Введений запит на мові запитів SPARQL

Результат запиту відображається на панелі зліва. У даному випадку це простий список класів (рисунок 5.31).

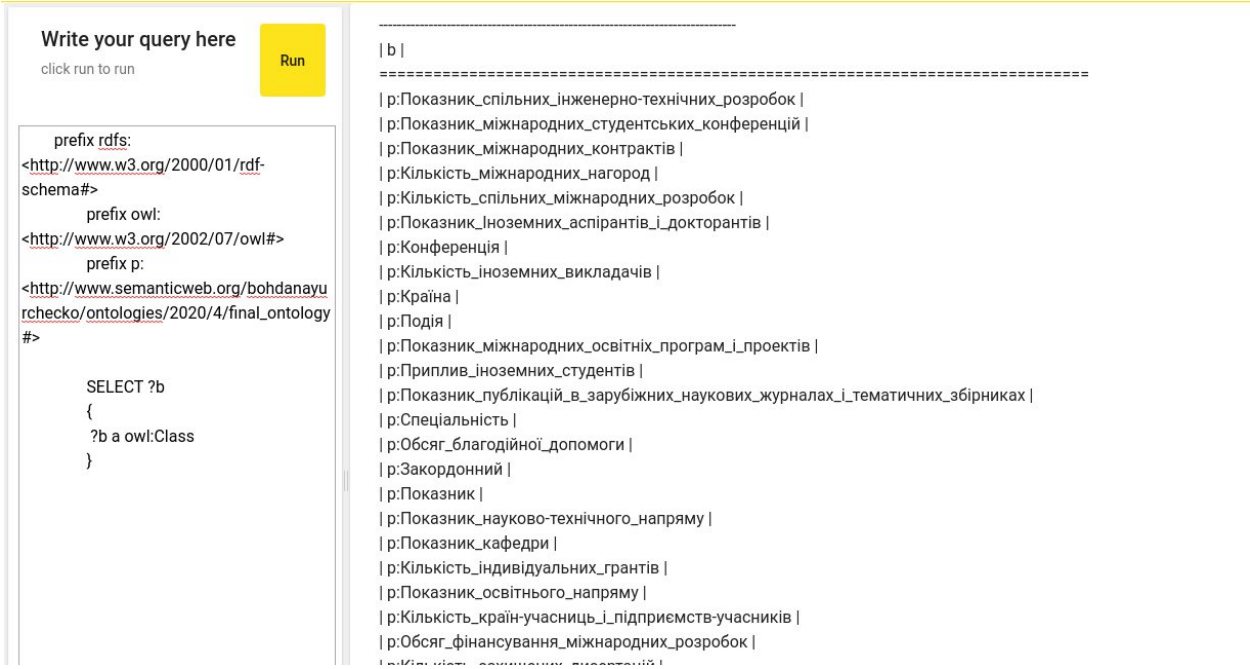


Рисунок 5.31 - Результати виконання запиту

У даному розділі був описаний спосіб роботи в системі і права користувачів із різними ролями. У системі є 4 основні сторінки, що дозволяють управляти онтологіями і їх версіями у системі, вводити дані, редагувати, видаляти та переглядати у різних режимах : у табличному за класами та за допомогою написання запитів на мові запитів - SPARQL.

ВИСНОВКИ

В рамках дипломної роботи було виконано такі завдання;

1. Було проаналізовано звіти про міжнародну діяльність за різні роки різних університетів.
2. Було визначено основні вимоги до інформаційної системи для аналізу міжнародного співробітництва.
3. Було розроблено онтологію із ієрархією класів та зв'язками між ними, що дозволить зберігати значення різних показників міжнародної діяльність за різні роки для різних університетів та їх підрозділів.
4. Було спроектовано і розроблено веб-додаток для редагування і перегляду онтології для користувачів, що не мають досвіду роботи із професійними засобами розробки онтологій (наприклад, такими як Protege).
5. Система дозволяє вводити дані у онтологію (створювати нові об'єкти, редагувати та видаляти старі). Система дозволяє переглядати дані у двох режимах : табличному і за допомогою написання SPARQL запитів.
6. Система дозволяє виконувати SPARQL запити.
7. Система дозволяє працювати із декількома онтологіями.

Розроблений веб-додаток є односторінковим. Також розроблений продукт має реалізацію різних рівнів доступу для різних користувачів із різними ролями.

Розроблена система поставляється у вигляді 3 образів для Docker контейнерів. Система є платформонезалежною і розширюваною. У роботі наведено приклад конфігурації контейнеру для хостингу sloppy.io.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. С.І.Сидоренко. Міжнародна діяльність університету: наближаючись до європейських стандартів [2013] [Електронний ресурс] / С.І.Сидоренко. – 2014. – Режим доступу до ресурсу: https://kpi.ua/links-13.

2. Підсумки міжнародної діяльності університету в 2017 році та завдання на 2018 рік. Зі звіту проректора з міжнародних зв'язків С.І. Сидоренка про роботу в 2017 р. [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://kpi.ua/2017-dms.

3. Програма Міжнародного Співробітництва Київського Міжнародного Унівреситету [Електронний ресурс] // ПЗВО "Київський Міжнародний Університет". – 2019. – Режим доступу до ресурсу: https://kymu.edu.ua/upload/pdf_files/prohrama_mizhnarodnoho_spivrobitnytstva.pdf.

4. Свєженцева О.І. «ЗВІТ ПРО МІЖНАРОДНУ ДІЯЛЬНІСТЬ ДВНЗ «УЖНУ» У 2016 РОЦІ» [Електронний ресурс] / Свєженцева О.І. – 2016. – Режим доступу до ресурсу: https://www.uzhnu.edu.ua/en/infocentre/get/11203.

5. Міжнародна діяльність [Електронний ресурс] – Режим доступу до ресурсу: http://www.ubs.edu.ua/images/2017/mizhnarodna_diyalnist/ZvitMD.pdf.

6. Документація. Securing Applications and Services Guide [Електронний ресурс] – Режим доступу до ресурсу: https://www.keycloak.org/docs/latest/securing_apps/index.html.

7. Уоллс, Крейг Spring в действии / Крейг Уоллс. - М.: ДМК Пресс, 2015. - 754 с.

8. David Recordon. OpenID 2.0: a platform for user-centric identity management / David Recordon, Drummond Reed. // DIM '06: Proceedings of the second ACM workshop on Digital identity management. – 2006. – С. 11–16.

9. DuCharme B. Learning SPARQL, 2nd Edition / Bob DuCharme., 2013. – 386 с.

10. Mouat A. Using Docker: Developing and Deploying Software with Containers / Adrian Mouat., 2015. – 354 с.

11. Freeman A. Pro Angular 6 / Adam Freeman., 2018. – 804 с.

12. Bloch J. Effective Java (3rd Edition) / Joshua Bloch., 2017. – 392 с.

Додаток 1

Онтологічна інформаційна система моніторингу показників рівня міжнародного співробітництва

Специфікація

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТВ6149_20Б

Аркушів 2

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 12-1	Текст програмного модулю	com/bohdanayurchenko/ontologyapp/service/
УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 12-2	Вихідний код модулів	.zip файл
УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 12-3	com/bohdanayurchenko/ontologyapp/model/*.java	Програмний код
УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 12-4	com/bohdanayurchenko/ontologyapp/controller/*.java	Програмний код
УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 12-5	ontology.owl	Онтологія
УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 13-1	Опис програми	

Додаток 2

Онтологічна інформаційна система моніторингу показників рівня міжнародного співробітництва

Текст програмного модулю

УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 12-1

Аркушів 10

2020

```
package com.bohdanayurchenko.ontologyapp.service;

import com.bohdanayurchenko.ontologyapp.exceptions.DoesNotExistException;
import com.bohdanayurchenko.ontologyapp.model.OntologyClassDTO;
import com.bohdanayurchenko.ontologyapp.model.OntologyClassDTOWithProperties;
import com.bohdanayurchenko.ontologyapp.model.properties.OntologyPropertyODT;
import com.bohdanayurchenko.ontologyapp.repository.LocalFileModelRepo;
import lombok.extern.log4j.Log4j2;
import org.apache.jena.ontology.*;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;

@Service
@Log4j2
public class OntologyClassService {
  @Autowired
  private LocalFileModelRepo fileModelRepo;
  @Autowired
  private OntologyInstanceService ontologyInstanceService;

  public OntologyClassService() {
  }

  /**
   * retrieves information about ontology class in the ontology
   * @param uri uri of the class in the ontology
   * @return class dto with main information
   * @throws DoesNotExistException in case if class doesn't exist
   */
  public OntologyClassDTO getOntologyClass(String uri) throws DoesNotExistException {
    log.debug("uri : '" + uri + "'");
    OntModel model = fileModelRepo.getModel();
```

```
    OntClass ontClass = model.getOntClass(uri);
    return OntologyClassDTO.ofClass(ontClass);
  }

  /**
   * retrieves information of ontology class with detailed information about its properties
   * @param uri uri of the class in the ontology
   * @return class dto with properties dtos
   * @throws DoesNotExistException in case if class doesn't exist
   */
  public OntologyClassDTOWithProperties getOntologyClassWithProperties(String uri) throws
DoesNotExistException {
    log.debug("uri : '" + uri + "'");
    OntModel model = fileModelRepo.getModel();
    OntClass ontClass = model.getOntClass(uri);
    List<OntologyPropertyODT> properties = getOntologyPropertyDTOsForClass(model, ontClass);
    return OntologyClassDTOWithProperties.ofClassWIthProperties(ontClass, properties);
  }

  private List<OntologyPropertyODT> getOntologyPropertyDTOsForClass(OntModel model, OntClass ontClass)
{
    List<OntologyPropertyODT> properties = new ArrayList<>();
    ExtendedIterator<DatatypeProperty> ontPropertyExtendedIterator = model.listDatatypeProperties();
    while (ontPropertyExtendedIterator.hasNext()) {
      DatatypeProperty essaProperty = ontPropertyExtendedIterator.next();
      if (essaProperty.hasDomain(ontClass))
        properties.add(OntologyPropertyODT.of(essaProperty));
    }
    ExtendedIterator<ObjectProperty> ontPropertyExtendedIterator2 = model.listObjectProperties();
    while (ontPropertyExtendedIterator2.hasNext()) {
      ObjectProperty essaProperty = ontPropertyExtendedIterator2.next();
      if (essaProperty.hasDomain(ontClass))
        properties.add(OntologyPropertyODT.of(essaProperty));
    }
    ExtendedIterator<OntProperty> ontPropertyExtendedIterator3 = ontClass.listDeclaredProperties();
    while (ontPropertyExtendedIterator3.hasNext()) {
      OntProperty essaProperty = ontPropertyExtendedIterator3.next();
      OntologyPropertyODT of = OntologyPropertyODT.of(essaProperty);
```

```
      if (!properties.contains(of)) {
        properties.add(of);
      }
    }
    for (OntologyPropertyODT property : properties) {
      property.setOptions(ontologyInstanceService.getRangeValuesForObjectProperty(property.getUri()));
    }
    log.debug("listed properties  " + properties);
    return properties;
  }

  /**
   * retrieves properties for ontology class
   * @param model ontology model
   * @param ontClass class
   * @return list of ontology properties
   */
  public List<OntProperty> getOntologyPropertiesForClass(OntModel model, OntClass ontClass) {
    List<OntProperty> properties = new ArrayList<>();
    ExtendedIterator<DatatypeProperty> ontPropertyExtendedIterator = model.listDatatypeProperties();
    while (ontPropertyExtendedIterator.hasNext()) {
      DatatypeProperty essaProperty = ontPropertyExtendedIterator.next();
      if (essaProperty.hasDomain(ontClass))
        properties.add(essaProperty);
    }
    ExtendedIterator<ObjectProperty> ontPropertyExtendedIterator2 = model.listObjectProperties();
    while (ontPropertyExtendedIterator2.hasNext()) {
      ObjectProperty essaProperty = ontPropertyExtendedIterator2.next();
      if (essaProperty.hasDomain(ontClass))
        properties.add(essaProperty);
    }
    ExtendedIterator<OntProperty> ontPropertyExtendedIterator3 = ontClass.listDeclaredProperties();
    while (ontPropertyExtendedIterator3.hasNext()) {
      OntProperty essaProperty = ontPropertyExtendedIterator3.next();
      if (!properties.contains(essaProperty)) {
        properties.add(essaProperty);
      }
    }
```

```
    log.debug("listed properties  " + properties);
    return properties;
  }

  /**
   * retrieves all classes in the ontology
   * @return all classes in the ontology
   */
  public Collection<OntologyClassDTO> getOntologyClasses() {
    List<OntologyClassDTO> rootClasses = new ArrayList<>();
    OntModel model = fileModelRepo.getModel();
    ExtendedIterator<OntClass> classes = model.listHierarchyRootClasses();
    while (classes.hasNext()) {
      OntClass thisClass = classes.next();
      OntologyClassDTO e = OntologyClassDTO.ofClassWIthChildren(thisClass);
      if (e != null) {
        log.debug("adding to the root class : " + e);
        rootClasses.add(e);
      }
    }
    return Collections.singletonList(new OntologyClassDTO("", "Root", rootClasses));
  }

}
package com.bohdanayurchenko.ontologyapp.service;

import com.bohdanayurchenko.ontologyapp.model.OntologyInstanceImplImpl;
import com.bohdanayurchenko.ontologyapp.model.properties.PropertyAndValue;
import com.bohdanayurchenko.ontologyapp.repository.LocalFileModelRepo;
import com.bohdanayurchenko.ontologyapp.exceptions.DoesNotExistException;
import com.bohdanayurchenko.ontologyapp.model.OntologyBaseDTO;
import com.bohdanayurchenko.ontologyapp.model.OntologyInstanceImpl;
import lombok.extern.log4j.Log4j2;
import org.apache.jena.ontology.*;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.validation.Valid;
import java.util.*;

@Service
@Log4j2
public class OntologyInstanceService {

  @Autowired
  private LocalFileModelRepo fileModelRepo;
  @Autowired
  private OntologyClassService ontologyClassService;

  public OntologyInstanceService() {
  }

  /**
   * retrives individual information with properties that are binded to the particular class
   * @param individualUri uri of individual
   * @param classUri uri of a class that is the type of the instance
   * @return instance dto with values of all properties
   * @throws DoesNotExistException if instance doesn't exist
   */
  public OntologyInstanceImplImpl getOntologyInstance(String individualUri, String classUri) throws
DoesNotExistException {
    OntModel model = fileModelRepo.getModel();
    return new OntologyInstanceImplImpl(model.getIndividual(individualUri),
        ontologyClassService.getOntologyPropertiesForClass(model, model.getOntClass(classUri)));
  }

  /**
   * retrieves all instances of the class from the ontology
   * @param classUri uri of a class
   * @return list of instances
```

```
   */
  public Collection<OntologyInstanceImplImpl> getInstancesByClass(String classUri) {
    OntModel model = fileModelRepo.getModel();
    OntClass ontClass = model.getOntClass(classUri);
    Set<OntologyInstanceImplImpl> instances = new HashSet<>();
    List<OntProperty> ontologyPropertiesForClass =
ontologyClassService.getOntologyPropertiesForClass(model, model.getOntClass(classUri));
    ExtendedIterator<? extends OntResource> extendedIterator = ontClass.listInstances();
    while (extendedIterator.hasNext()) {
      OntResource next = extendedIterator.next();
      if (next.isIndividual()) {
        OntologyInstanceImplImpl of = new OntologyInstanceImplImpl(next.asIndividual(),
ontologyPropertiesForClass);
        of.setClassUri(classUri);
        instances.add(of);
      }
    }
    instances.remove(null);
    return instances;
  }



  /**
   * retrieves all instances from the ontology
   * @return list of instances
   */
  public Collection<? extends OntologyBaseDTO> getInstances() {
    OntModel model = fileModelRepo.getModel();
    Set<OntologyBaseDTO> instances = new HashSet<>();
    ExtendedIterator<? extends OntResource> extendedIterator = model.listIndividuals();
    while (extendedIterator.hasNext()) {
      instances.add(OntologyBaseDTO.of(extendedIterator.next()));
    }
    instances.remove(null);
    return instances;
  }
```

```
  /**
   * updates instance in the ontology
   * @param ontologyInstance instance with new values
   * @return updated instance
   * @throws DoesNotExistException in case if something doesn't exist
   */
  public OntologyBaseDTO updateInstance(@Valid OntologyInstanceImplImpl ontologyInstance) throws
DoesNotExistException {
    this.deleteInstance(ontologyInstance.getUri());
    return this.createInstance(ontologyInstance);
  }

  /**
   * creates instance with given values
   * @param ontologyInstance ontology instance with all values
   * @return newly created instance
   */
  public OntologyBaseDTO createInstance(OntologyInstanceImplImpl ontologyInstance) {
    log.debug(ontologyInstance);
    OntModel model = fileModelRepo.getModel();
    OntClass classResource = model.getOntClass(ontologyInstance.getClassUri());
    Individual individual = classResource.createIndividual(classResource.getNameSpace() +
ontologyInstance.getLocalName());
    OntologyBaseDTO of = OntologyInstanceImpl.of(individual);
    for (PropertyAndValue propertyAndValue : ontologyInstance.getPropertyAndValue()) {
      log.debug(propertyAndValue);
      log.debug(propertyAndValue.getPropertyUri());
      log.debug(propertyAndValue.getValue());
      for (String value : propertyAndValue.getValue()) {
        OntProperty property = model.getOntProperty(propertyAndValue.getPropertyUri());
        if (property.isDatatypeProperty()) {
          log.debug("data");
          individual.addLiteral(property, value);
        } else {
          log.debug("object");
          individual.addProperty(property, model.getResource(classResource.getNameSpace() + value));
        }
```

```
        }
      }
      log.debug("before error");
      fileModelRepo.writeModel(model);
      return of;
    }

    /**
     * deletes instance with given uri
     * @param uri uri of an instance to delete
     * @return true in case of success
     */
    public Boolean deleteInstance(String uri) {
      OntModel model = fileModelRepo.getModel();
      Individual individual = model.getIndividual(uri);
      individual.remove();
      fileModelRepo.writeModel(model);
      return true;
    }

    /**
     * retrieves all possible values of object property
     * @param propertyId property uri to get values
     * @return collection of values
     */
    public Collection<String> getRangeValuesForObjectProperty(String propertyId) {
      List<String> validValues = new ArrayList<>();
      OntModel model = fileModelRepo.getModel();
      OntProperty ontProperty = model.getOntProperty(propertyId);
      ExtendedIterator<? extends OntResource> extendedIterator = ontProperty.listRange();
      while (extendedIterator.hasNext()) {
        ExtendedIterator<? extends OntResource> extendedIterator1 =
model.getOntClass(extendedIterator.next().getURI()).listInstances();
        while ((extendedIterator1.hasNext())) {
          validValues.add(extendedIterator1.next().asIndividual().getLocalName());
        }
      }
```

```
    log.debug(validValues);
    return validValues;
  }

  /**
   * returns all direct instances of a class (without subclasses)
   * @param classUri uri of a class
   * @return list of direct instances
   */
  public Collection<OntologyInstanceImplImpl> getDirectInstancesByClass(String classUri) {
    OntModel model = fileModelRepo.getModel();
    OntClass ontClass = model.getOntClass(classUri);
    Set<OntologyInstanceImplImpl> instances = new HashSet<>();
    List<OntProperty> ontologyPropertiesForClass =
ontologyClassService.getOntologyPropertiesForClass(model, model.getOntClass(classUri));
    ExtendedIterator<? extends OntResource> extendedIterator = ontClass.listInstances(true);
    while (extendedIterator.hasNext()) {
      OntResource next = extendedIterator.next();
      if (next.isIndividual()) {
        OntologyInstanceImplImpl of = new OntologyInstanceImplImpl(next.asIndividual(),
ontologyPropertiesForClass);
        of.setClassUri(classUri);
        instances.add(of);
      }
    }
    instances.remove(null);
    return instances;
  }
}
```

Додаток 3

Онтологічна інформаційна система моніторингу показників рівня міжнародного співробітництва

Опис програмного модулю

УКР.НТУУ “КПІ ім. Ігоря Сікорського ”.ТВ6149_20Б 13-1

Аркушів 5

2020

АНОТАЦІЯ

Метою роботи було створення інформаційної системи для моніторингу показників рівня міжнародного співробітництва.

Модуль був розроблений у середовищі Intellij IDEA на мові програмування Java із використанням фреймворку Spring Boot та бібліотекою для роботи із онтологіями - Apache Jena.

Даний модуль виконує головні функції роботи із онтологіями : пошук, створення, редагування та видалення різних сутностей із онтології.

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

Модулі були написані на мовах програмування Java та JavaScript, для реалізації безпеки, авторизації було використано готове рішення Keycloak.

Додаток надає можливість вводити, редагувати та переглядати значення показників міжнародного співробітництва у табличному режимі і за допомогою SPARQL запитів. Реалізація системі із використанням онтології в якості сховища даних робить систему гнучкою. Це означає, що в майбутньому, у разі зміни показників чи додаванні нових, систему буде дуже легко адаптувати до змін.

Гарний інтерфейс та архітектура веб-додатку роблять розроблену систему зручною для використання.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Було розроблено веб-додаток, що дозволить вводити, редагувати та переглядати значення різних показників міжнародного співробітництва.

Програмне рішення складається із 3 сервісів. Головний сервіс складається із пакетів: config, constroller, exceptions, model, repository, service.

Модуль service безпосередньо виконує усю роботу, звязану із онтологіями.

Клас OntologyClassService виконує такі функції:

- пошук усіх класів онтології;
- пошук одного класу в онтології;
- пошук класу в онтології та його властивостей.

Клас OntologyInstanceService виконує такі функції:

- пошук усіх інстансів в онтології ;
- пошук усіх безпосередніх інстансів одного класу;
- пошук усіх інстансів класу та його нащадків;
- пошук значень усіх властивостей інстансу;
- створення інстансу класу;
- редагування інстансу класу;
- видалення інстансу класу.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Логічна структура програми складається із 3 модулів, що є незалежними. Через таку архітектуру програма може легко масштабуватися.

Модулі написані із використанням сучасних фреймворків із дотриманням головних принципів програмування. Тому рішення у майбутньому буле легко розширюватися і підтримуватися.

4 ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Даний програмний модуль розроблено у середовищі IntelliJ IDEA та WebStrom - розроблені JetBrains. Були використані такі мови програмування як Java, JavaScript та такі фрейморки як Spring Boot, Angular 8 , Type Script та бібліотеку для роботи із онтологіями - Apache Jena.

Програмний модуль було протестовано в браузері Google Chrome 83.0.4103.61 на персональному комп’ютері, який працює на базі процесору Intel Core i7 1.80GHz та має 16 Гб оперативної пам’яті на операційній системі Fedora Workstation 32 та Xubuntu 20.

Розроблене програмне забезпечення є кросбраузереним та кросплатформенним та поставляється у вигляді образів для контейнерів Docker, що означає, що рішення дуже легко і просто встановити і почати користуватися.

5 ВИКЛИК І ЗАВАНТАЖЕННЯ

Для функціонування програми пострібно встановити і налаштувати Docker або Podman та Docker Compose. Після цього потрібно перейти в директорію із проектом і запустити програмний продукт командою “docker-compose up”.

Для початку роботи треба відкрити сайт у браузері.

6 ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для системи є файл онтології із розробленими класами у форматі .owl, значення показників міжнародного співробітництва та SPARQL запит.

Вихідними даними є результати SPARQL запиту та оновлена онтологія.

Додаток 4

Онтологічна інформаційна система моніторингу показників рівня міжнародного співробітництва

АПРОБАЦІЯ

XVIII міжнародна науково-практична конференція молодих вчених та студентів «Сучасні проблеми наукового забезпечення енергетики» / XVIII International scientific and practical conference of young scientists and students "Modern problems of scientific support of power engineering"
Київ, 21-24 квітня 2020 р.

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТВ6149_20Б

Аркушів 4

2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVIII Міжнародної науково-практичної конференції молодих вчених і студентів 2020 року

ТОМ 2

Київ- 2020

УДК 004.048

Студент 4 курсу, гр. ТВ-61 Юрченко Б.О.
Ст.викл. Дацюк О.А.

АНАЛІЗ СУЧАСНИХ РЕДАКТОРІВ ОНТОЛОГІЙ

Для того, аби розробка онтологій була простою – треба мати зручний, гарний і надійний редактор. На ринку зараз є велика кількість різних редакторів [1]: десктоп і онлайн редактори, деякі дозволяють лише відображати онтолігії, інші мають гарний графічний інтерфейс для редагування, деякі дозволяють працювати із колегами над одним проектом. Серед усього різноманіття доволі важко обрати саме те, що треба розробнику-початківцю.

Тому мета даного дослідження – систематизувати знання про функціонал і інші параметри доступних для розробника різних засобів розробки і відображення онтологій [2].

Найважливішими параметрами є: платформа; ціна; можливість редагування; можливість відображення; дизайн; підтримка; набір функцій тощо[3].

В таблиці наведено основні характеристики деяких редакторів онтологій.

Назва	Платформа	Безкоштовно	Візуалізація	Редагування	Відкритий код	Розробник	Розширення	Останній реліз
Protege	Online і Для ПК (Linux, Windows, MacOS,platform independent)	+	+	+	+	Stanford Center for Biomedical Informatics Research	+	desktop: 5.5 14/03/2019 web:4.0 Beta-2 12/08/2019
NeOn toolkit	Розширення для Eclipse (MacOS, Windows, Linux)	+	+	+	+	NeOn Project	+	2.5.2 - 12/12/2011.
Swoop	Для ПК (будь-яка ОС із JRE 1.4+)	+	+	+	+	MINDSWAP University of Maryland, College Park	+	04/01/2006
TopBraid Composer	Для ПК (Windows MacOS, Linux)	безкоштовна версія	+	+	-	TopQuadrant	+	free edition : 6.0.1 27/06/2019
Owlgred	Online і для ПК (Windows)	+	+	+ (версія ПК)	-	IMCS UL	+	1.6.10
FLuent Editor	Для ПК (Windows)	в деяких випадках	+	+	-	Cognitum	+	3.6.10.28710 01/12/2016
VocBench	Web based (будь-яка ОС із JRE 8+)	+	+	+	+	ART Group	+	VB3 7.0.0. 18/02/2020

Підчас дослідження було розглянуто декілька найбільш популярних редакторів.Усі розглянуті рішення є безкоштовними (або мають безкоштовну ліцензію в деяких випадках), і багато є мультипрлатформенними. Тому у кожного розробника онтологій є свобода вибору інструменту на основі інших важливих для нього параметрів.

Перелік посилань:

1. Ontology editors. URL: https://www.w3.org/wiki/Ontology_editors (дата звернення: 09.03.2020)

2. L.Globa, R.Novogrudska, A.Koval, V.Senchenko (March 8th 2018). Ontology for Application Development, DOI: 10.5772/intechopen.74042. URL: https://www.intechopen.com/books/ontology-in-information-science/ontology-for-application-development

3. Raimond, Yves & Abdallah, Samer & Sandler, Mark & Giasson, Frederick. (2007). The Music Ontology. Proceedings of the 8th International Conference on Music Information Retrieval, ISMIR 2007.